A Novel Learning Algorithm for Büchi Automata based on Family of DFAs and Classification Trees

Yong Li[a,b], Yu-Fang Chen[c], Lijun Zhang[a,d,b,*], Depeng Liu[a,b]

[a] State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing 100190
[b] University of Chinese Academy of Sciences, Beijing 100049
[c] Institute of Information Science, Academia Sinica, Taipei 11529
[d] Institute of Intelligent Software, Guangzhou, Beijing 511400

Abstract

In this paper, we propose a novel algorithm to learn a Büchi automaton from a teacher who knows an ω-regular language. The learned Büchi automaton can be a nondeterministic Büchi automaton or a limit deterministic Büchi automaton. The learning algorithm is based on learning a formalism called *family of DFAs* (FDFAs) recently proposed by Angluin and Fisman [1]. The main catch is that we use a *classification tree* structure instead of the standard *observation table* structure. The worst case storage space required by our algorithm is quadratically better than that required by the table-based algorithm proposed in [1]. We implement the proposed learning algorithms in the learning library ROLL (Regular Omega Language Learning), which also consists of other complete ω-regular learning algorithms available in the literature. Experimental results show that our tree-based learning algorithms have the best performance among others regarding the number of solved learning tasks.

Keywords: Büchi Automata, Learning Algorithm, Automata Learning, Observation Table, Family of DFAs, Classification Tree, L^*

1. Introduction

In the past two decades, learning-based automata inference techniques [2–5] have received significant attention from the community of formal verification. In general, the primary applications of automata learning techniques in the community can be categorized into two: *improving the efficiency and the scalability of verification techniques* [6–13] and *synthesizing abstract system models for further analysis* [14–23].

The former is usually based on the so called *assume-guarantee* compositional verification approach, which divides a verification task into several subtasks via a composition rule. Learning algorithms are applied to construct environmental assumptions of components in the rule automatically. For the latter, automata learning algorithms have been used to automatically generate interface models of computer programs [17–20, 24], to extract a model of system error

[*] Corresponding author

Email addresses: liyong@ios.ac.cn (Yong Li), yfc@iis.sinica.edu.tw (Yu-Fang Chen), zhanglj@ios.ac.cn (Lijun Zhang), liudp@ios.ac.cn (Depeng Liu)

traces for diagnosis purpose [22], to get a behavior model of programs for statistical program analysis [23], and to do model-based testing and verification [14–16]. Later, Vaandrager [25] explained the concept of *model learning* used in above applications. In particular, there are some robust libraries for finite automata learning available in the literature, e.g., libalf [26] and LearnLib [27].

Besides the classical finite automata learning algorithms, people have also developed and applied learning algorithms for richer models for the above two applications. For example, learning algorithms for register automata [28, 29] have been developed and applied to synthesizing program interface models. For timed automata, learning algorithms have been developed to automate the compositional verification of timed systems [10] and to verify specifications of the TCP protocol [30]. However, all the above results are for checking *safety properties* or synthesizing *finite behavior models* of systems/programs. Büchi automata are a standard model for describing liveness properties of distributed systems [31] and have been widely applied in the automata-based model checking framework [32] to describe properties to be verified as well as in the synthesis of reactive systems [33]. Moreover, Büchi automata have been used as a means to prove the termination of programs [34]. Therefore, in order to verify whether a system satisfies a liveness property with learning algorithms, a learning algorithm for Büchi automata can be employed.

Motivated by that, Maler and Pnueli introduced in [35] the first learning algorithm for Büchi automata, which is, however, only able to learn a strict subclass of ω-regular languages. The first learning algorithm of Büchi automata accepting the complete class of ω-regular languages was described in [36], based on the L^* algorithm [4] and the result of [37]. However, unlike the case for the finite automata learning, the research on applying Büchi learning algorithms for verification problems is still in its infancy despite the popularity of Büchi automata in the community.

One reason why the learning algorithms for Büchi automata have seldom been used is that the learning algorithms for Büchi automata are currently not efficient enough for model checking. Recently, Angluin and Fisman proposed a learning algorithm in [1] by learning a formalism called *family of DFAs* (FDFAs), based on the results of [38]. The main barrier of applying their learning algorithm in the verification is that their algorithm requires a teacher for FDFAs. To the best of our knowledge, FDFAs have not yet been applied in the verification while Büchi automata have already been used in several areas such as program termination analysis [39] and probabilistic verification [40]. As a main contribution, in this paper, we show that the FDFA learning algorithm in [1] can be adapted to support Büchi automata teachers.

To further improve the efficiency of Büchi learning algorithms, in this paper we propose a novel learning algorithm of Büchi automata accepting the complete class of ω-regular languages based on FDFAs and a *classification tree* structure (inspired by the tree-based L^* algorithm in [3] and the TTT learning algorithm in [41]). In terms of worst case storage space, the space required by our algorithm is quadratically better than that of the table-based algorithm proposed in [1]. We implement our learning algorithm for Büchi automata in the library ROLL [42](Regular Omega Language Learning, `http://iscasmc.ios.ac.cn/roll`), which includes all other Büchi automata learning algorithms of the complete class of ω-regular languages available in the literature. We compare the performance of those algorithms using a benchmark of 295 Büchi automata corresponding to all 295 LTL specifications available in Büchi Store [43], as well as 20 Büchi automata whose languages cannot be specified by LTL formulas. Experimental results show that our tree-based algorithms have the best performance among others regarding the number of solved learning tasks.

To summarize, our contribution includes the following. (1) Adapting the algorithm in [1] to support Büchi automata teachers. (2) A novel Büchi automata learning algorithm for the complete class of ω-regular languages based on FDFAs and classification trees. (3) A comprehensive empirical evaluation of all the Büchi automata learning algorithms available in the literature with ROLL.

A previous version of our learning algorithm appeared in [44]. Compared to the previous version, we have added more examples and intuitions about the proposed learning algorithms. For instance, we have added Fig. 2 in order to give the readers an idea of three different types of canonical FDFAs. We have provided detailed proofs and complexity analysis. Many proofs given here are not trivial so we add them in the hope that the reader may benefit from those ideas in their own works.

Another contribution made in this paper is that we extend the learning algorithm for Büchi automata proposed in [44] to a learning algorithm for *limit deterministic* Büchi automata. Limit deterministic Büchi automata are a new variety of Büchi automata introduced in [40, 45] for qualitative verification of Markov Decision Processes (MDPs). More precisely, our learned limit deterministic Büchi automata have two components, namely the *initial* component and the *accepting* component where two components are both deterministic and all accepting states are contained in the accepting component. The nondeterminism only occurs on the transitions from the initial component to the accepting component. We are aware that the same Büchi automata are also defined in [46]. Moreover, limit deterministic Büchi automata are widely used in the program termination analysis according to [39, 47]. Therefore, it is intriguing to see whether we can apply our learning algorithm in probabilistic verification and program analysis and we leave this to future work.

2. Preliminaries

Let $\oplus$ be the standard modular arithmetic operator. Let A and B be two sets. We use $A \ominus B$ to denote their *symmetric difference*, i.e., the set $(A \setminus B) \cup (B \setminus A)$. We use $[i \cdots j]$ to denote the set $\{i, i+1, \ldots, j\}$. Let Σ be a finite non-empty set of letters called *alphabet*. A *word* is a finite or infinite sequence $w = w[1]w[2]\cdots$ of letters in Σ. We use ϵ to represent an empty word. The set of all finite words is denoted by Σ^*, and the set of all infinite words, called ω-words, is denoted by Σ^ω. Moreover, we also denote by Σ^+ the set $\Sigma^* \setminus \{\epsilon\}$. We use $|u|$ to denote the length of the finite word u. We denote by $w[i]$ the i-th letter of a word w. We use $w[i \cdots k]$ to denote the subword of w starting at the i-th letter and ending at the k-th letter, inclusive, when $i \leq k$ and the empty word ϵ when $i > k$. Given a finite word $u = u[1]\cdots u[k]$ and a word w, we denote by $u \cdot w$ the *concatenation* of u and w, i.e., the finite or infinite word $u \cdot w = u[1]\cdots u[k]w[1]\cdots$, respectively. We may just write uw instead of $u \cdot w$.

A *finite automaton* (FA) is a tuple $A = (\Sigma, Q, q_0, \delta, F)$ consisting of a finite alphabet Σ, a finite set Q of states, an initial state q_0, a set $F \subseteq Q$ of accepting states, and a transition relation $\delta \subseteq Q \times \Sigma \times Q$. For convenience, we also use $\delta(q, a)$ to denote the set $\{q' \mid (q, a, q') \in \delta\}$. A *run* of an FA on a finite word $u = a_1 a_2 a_3 \cdots a_n$ is a sequence of states $q_0, q_1, \ldots, q_n$ such that q_0 is the initial state and $(q_i, a_{i+1}, q_{i+1}) \in \delta$ for every $0 \leq i < n$ where $n \geq 1$. The run on u of an FA A is *accepting* if $q_n \in F$. A word u is accepted by an FA A if A has an accepting run on it. A *finite language* is a subset of Σ^*; the language of an FA A, denoted by $L(A)$, is the set $\{u \in \Sigma^* \mid u \text{ is accepted by } A\}$; the language of an FA is called a *regular* language. Let A and B be two FAs; one can construct a product FA $A \times B$ accepting $L(A) \cap L(B)$ using a standard product construction (see for example [48, Theorem 4.8]).

A *deterministic finite automaton* (DFA) is an FA such that $\delta(q, a)$ is a singleton for any $q \in Q$ and $a \in \Sigma$. For DFAs, we write $\delta(q, a) = q'$ instead of $\delta(q, a) = \{q'\}$. The transition can be lifted to words by defining $\delta(q, \epsilon) = q$ and $\delta(q, av) = \delta(\delta(q, a), v)$ for each $q \in Q, a \in \Sigma$ and $v \in \Sigma^*$. We also use $A(v)$ as a shorthand for $\delta(q_0, v)$.

An *ω-language* is a subset of Σ^ω. Words of the form uv^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$, are called *ultimately periodic* words. We use a pair of finite words (u, v) to denote the ultimately periodic word $w = uv^\omega$. We also call (u, v) a *decomposition* of w. Note that an ultimately periodic word uv^ω can have several decompositions: for instance (u, v), (uv, v) and (uvv, vv) are all valid decompositions of uv^ω. For an ω-language L, let $\mathrm{UP}(L) = \{uv^\omega \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega \in L\}$, i.e., all ultimately periodic words in L. In this paper, we are particularly interested in a class of ω-languages called *ω-regular* languages. In the following, we will introduce some automata representations of ω-regular languages and discuss their roles in ω-regular language learning.

3. Representations of ω-Regular Languages

The first representation of ω-regular languages introduced here is *Büchi automata*, which were originally introduced by Julius Richard Büchi in [49] and now have been widely used in model checking. A Büchi automaton (BA) has the same structure as an FA, except that it accepts only infinite words. A run of a BA on an infinite word is defined similarly to that of an FA except that instead of ending in a state, it visits an infinite sequence of states. An infinite word w is accepted by a BA A iff A has a run on w visiting an accepting state infinitely often. The language of a BA A, denoted by $L(A)$, is the set $\{w \in \Sigma^\omega \mid w \text{ is accepted by } A\}$. An ω-language $L \subseteq \Sigma^\omega$ is ω-regular iff there exists a BA A such that $L = L(A)$. A BA is a *deterministic Büchi automaton* (DBA) if $|\delta(q, a)| \leq 1$ for each $q \in Q$ and $a \in \Sigma$. A BA is a *limit deterministic Büchi automaton* (LDBA) if its states set Q can be partitioned into two disjoint sets $Q_N \cup Q_D$, such that 1) $\delta(q, a) \subseteq Q_D$ and $|\delta(q, a)| \leq 1$ for $q \in Q_D$ and $a \in \Sigma$ and 2) $F \subseteq Q_D$. Limit deterministic Büchi automata can also accept the complete class of ω-regular languages [40, 46] and thus have the same expressive power as BAs, while DBAs are strictly less expressive than BAs [50]. For an ω-regular language L, the set of ultimately periodic words of L, denoted by $\mathrm{UP}(L)$, can be seen as the fingerprint of L, as stated below.

Theorem 1 (Ultimately Periodic Words of ω-Regular Languages [37, 49]). *Let L, L' be two ω-regular languages. Then $L = L'$ if and only if $UP(L) = UP(L')$.*

Proof (Sketch). We first prove the fact that every non-empty ω-regular language L contains at least one ultimately periodic word. Let L be a non-empty ω-regular language and A be a Büchi automaton accepting L. Assume that A is defined by the tuple $(\Sigma, Q, q_0, \delta, F)$. There must be an ω-word $w \in L$ such that a corresponding run of A on w visits some accepting state $q_f \in F$ infinitely often. Define two FAs $A_1 = (\Sigma, Q, q_0, \delta, \{q_f\})$ and $A_2 = (\Sigma, Q, q_f, \delta, \{q_f\})$ and let $U = L(A_1)$ and $V = L(A_2)$. It immediately follows that $U \neq \emptyset$, $V \neq \emptyset$, and $w \in UV^\omega \subseteq L$. Therefore, we can find an ultimately periodic word $uv^\omega \in L$ with $u \in U$ and $v \in V$.

Now we are ready to give the proof of the theorem. The implication from the left to the right is trivial. It is easy to see that $L \ominus L' = (L \setminus L') \cup (L' \setminus L)$ is an ω-regular language since ω-regular languages are closed under the operations of intersection, union, and complementation [49]. Assume that $\mathrm{UP}(L) = \mathrm{UP}(L')$. It follows that $L \ominus L'$ does not contain any ultimately periodic words and thus $L \ominus L'$ is an empty language according to the fact that every non-empty ω-regular language L contains at least one ultimately periodic word. Therefore, we conclude that $L = L'$. ■

An immediate consequence of Theorem 1 is that, for every two ω-regular languages L_1 and L_2, if $L_1 \neq L_2$ then there must exist some ultimately periodic word $xy^{\omega} \in \mathrm{UP}(L_1) \ominus \mathrm{UP}(L_2)$. Therefore, Calbrix *et al.* proposed a special DFA as another representation of ω-regular languages in [37]. More precisely, they construct a DFA $D_{\$}$ from a BA A to represent $L = L(A)$ such that $L(D_{\$}) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^+, uv^{\omega} \in \mathrm{UP}(L)\}$ where $\$ \notin \Sigma$ is a fresh letter. Intuitively, $D_{\$}$ accepts every ultimately periodic word uv^{ω} of $\mathrm{UP}(L)$ by means of a decomposition represented as the finite word $u\$v$.

Our goal in this paper is to learn the ω-regular languages by means of Büchi automata and our idea of learning goes back to the learning algorithm L^* proposed by Angluin in [4]. In her seminal work [4], Angluin proposed to learn an unknown regular language R by means of a DFA and the *right congruence* is the theoretical foundation for it to discover states in the target DFA accepting R. A right congruence is an equivalence relation $\backsim$ on Σ^* such that $x \backsim y$ implies $xv \backsim yv$ for every $x, y, v \in \Sigma^*$. We denote by $|\backsim|$ the *index* of $\backsim$, i.e., the number of equivalence classes of $\backsim$. We use $\Sigma^*/_{\backsim}$ to denote the set of equivalence classes of the right congruence $\backsim$. A *finite right congruence* is a right congruence with a finite index. For a word $u \in \Sigma^*$, we denote by $[u]_{\backsim}$ the class of $\backsim$ in which u resides.

For a regular language R, there exists a *canonical* DFA accepting R with the minimum number of states where every state of that DFA corresponds to an equivalence class of the right congruence of R; such a DFA is unique when ignoring the names of states and is often called the *minimal* DFA of R [48]. Thus L^* is able to learn the minimal DFA of R by identifying all its states or equivalently equivalence classes of right congruences. The main obstacle in learning ω-regular languages by means of Büchi automata is that there is a lack of right congruences of canonical Büchi automata for the complete class of ω-regular languages [51]. The BA learning algorithm proposed in [36] circumvents the lack of right congruences by first using L^* algorithm to learn the DFA $D_{\$}$ as defined in [37], and then transforming $D_{\$}$ to a BA. Another way to bypass the obstacle is to define other kinds of right congruences for ω-regular languages. Inspired by the work of Arnold [52], Maler and Stager [53] proposed the notion of *family of right-congruences* (FORC for short) for ω-regular languages. Based on the idea of FORC, Angluin and Fisman [1] further proposed to learn ω-regular languages via a formalism called *family of DFAs*, in which every DFA corresponds to a right congruence with a finite index. The BA learning algorithm proposed in this paper first learns an FDFA and then transforms the learned FDFA to a BA.

Definition 1 (Family of DFAs (FDFA) [1]). A family of DFAs $\mathcal{F} = (M, \{A^q\})$ over an alphabet Σ consists of a leading DFA $M = (\Sigma, Q, q_0, \delta, \emptyset)$ and a progress DFA $A^q = (\Sigma, Q_q, s_q, \delta_q, F_q)$ for each $q \in Q$.

Notice that the leading DFA M is a DFA *without* accepting states. Each FDFA $\mathcal{F}$ characterizes a set of ultimately periodic words $\mathrm{UP}(\mathcal{F})$.

Definition 2 (Acceptance condition of FDFA). [1] An ultimately periodic word w is in $\mathrm{UP}(\mathcal{F})$ iff it has a decomposition (u, v) *accepted* by $\mathcal{F}$. A decomposition (u, v) is accepted by $\mathcal{F}$ iff $M(uv) = M(u)$ and $v \in L(A^{M(u)})$.

An example of an FDFA $\mathcal{F}$ is depicted in Fig. 1. M has only one state ϵ. The progress DFA of ϵ is A^{ϵ}. The word $(ba)^{\omega}$ is in $\mathrm{UP}(\mathcal{F})$ because it has a decomposition (ba, ba) such that

[1] We remark that our acceptance condition defined in Definition 2 is different from the original one in [1]. This difference, however, does not change the set of ultimately periodic words accepted by a canonical FDFA introduced later.

$M(ba \cdot ba) = M(ba)$ and $ba \in L(A^{M(ba)}) = L(A^{\epsilon})$. It is easy to see that the decomposition (bab, ab) is not accepted by $\mathcal{F}$ since $ab \notin L(A^{M(bab)}) = L(A^{\epsilon})$.

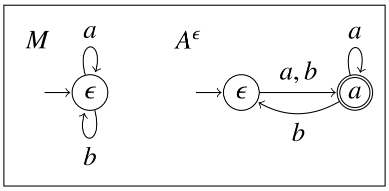


Figure 1: An example of an FDFA

For any ω-regular language L, there exists an FDFA $\mathcal{F}$ such that $\mathrm{UP}(L) = \mathrm{UP}(\mathcal{F})$ [1]. We show in Sect. 7 that an FDFA, however, does not necessarily accept an ω-regular language. In [1], three kinds of FDFAs are suggested as the canonical representations of ω-regular languages, namely *periodic FDFAs*, *syntactic FDFAs* and *recurrent FDFAs*. Their formal definitions are given below in terms of right congruences.

The right congruence $\backsim_L$ of a given ω-regular language L is defined such that $x \backsim_L y$ iff $\forall w \in \Sigma^{\omega}.xw \in L \Longleftrightarrow yw \in L$. The index of $\backsim_L$ is finite because it is not larger than the number of states in a *deterministic Muller automaton* (DMA) accepting L [38]. A DMA A has the same structure as a DBA except that it has a set of sets of states $AC \subseteq 2^Q$ rather than a set of accepting states $F \subseteq Q$ where Q is the set of states of A. Further, an ω-word w is accepted by a DMA A iff the set of infinitely appearing states in the run of A on w is exactly an element of AC.

Let $A = (\Sigma, Q, q_0, \delta)$ be a DFA by ignoring its accepting states. We define a right congruence $\backsim_A$ as follows: $x \backsim_A y$ iff $\delta(q_0, x) = \delta(q_0, y)$ for any $x, y \in \Sigma^*$. Let $\backsim_L$ be a right congruence of a language L and A a DFA and we say $\backsim_A$ and $\backsim_L$ are *consistent* if for every $x, y \in \Sigma^*$, $x \backsim_A y \Longleftrightarrow x \backsim_L y$.

In this paper, by an abuse of notation, we use a finite word u to denote the state in a DFA in which the equivalence class $[u]$ resides. The three types of canonical FDFAs introduced in [1] also follow the idea to recognize an ω-regular language L by means of $\mathrm{UP}(L)$ as $D_{\$}$ does in [37].

We first introduce the periodic FDFA, which Angluin and Fisman called in [1] the "FDFA version" of the language $L(D_{\$})$ defined in [37].

Definition 3 (Periodic FDFA [1]). Let L be an ω-regular language. Then the periodic FDFA $\mathcal{F} = (M, \{A^q\})$ of L is defined as follows.
The leading DFA M is the tuple $(\Sigma, \Sigma^*/_{\backsim_L}, [\epsilon]_{\backsim_L}, \delta, \emptyset)$, where $\delta([u]_{\backsim_L}, a) = [ua]_{\backsim_L}$ for all $u \in \Sigma^*$ and $a \in \Sigma$.

We define the periodic right congruence $\approx_P^u$ for each progress DFA A^u of $\mathcal{F}$ as follows: $x \approx_P^u y$ iff $\forall v \in \Sigma^*.u(xv)^{\omega} \in L \Longleftrightarrow u(yv)^{\omega} \in L$.

The progress DFA A^u of the state $[u]_{\backsim_L} \in \Sigma^*/_{\backsim_L}$ is the tuple $(\Sigma, \Sigma^*/_{\approx_P^u}, [\epsilon]_{\approx_P^u}, \delta_P, F_P)$, where we have that $\delta_P([v]_{\approx_P^u}, a) = [va]_{\approx_P^u}$ for all $v \in \Sigma^*$ and $a \in \Sigma$. The set of accepting states F_P is the set of equivalence classes $[v]_{\approx_P^u}$ for which $uv^{\omega} \in L$.

It has been shown in [1] that for any $u, x, y, v \in \Sigma^*$, $xv \approx_P^u yv$ holds if $x \approx_P^u y$ holds and $\approx_P^u$ is of finite index for any given ω-regular language L. Therefore $\approx_P^u$ is a right congruence of finite index for each $u \in \Sigma^*$ so one can build a finite transition system from each of them. Note that the syntactic right congruences and the recurrent right congruences introduced later are also of finite index.

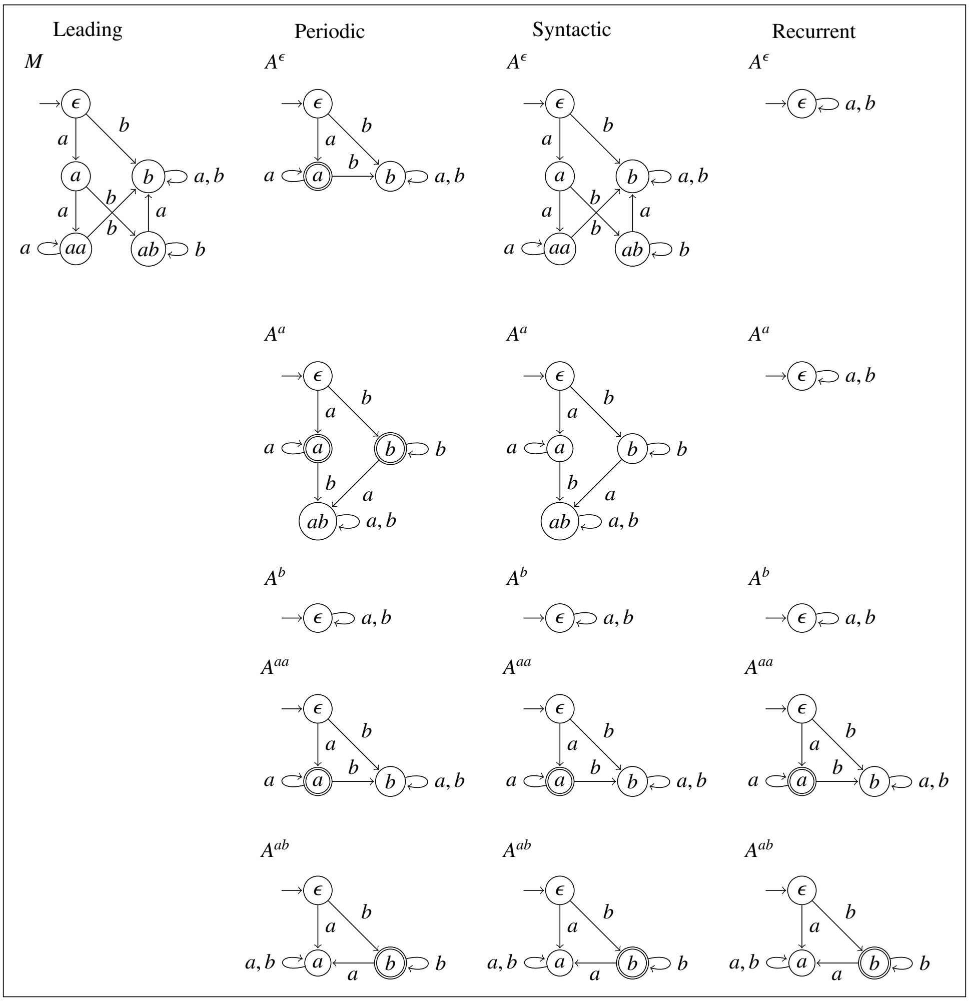


Figure 2: An example of three types of canonical FDFAs $\mathcal{F} = (M, \{A^q\})$ of $L = a^\omega + ab^\omega$

To further explain the canonical FDFAs, we introduce another notion for FDFAs. We say a decomposition (u, v) is *captured* by an FDFA $\mathcal{F} = (M, \{A^q\})$ if $A^{M(u)}(v)$ is an accepting state of $A^{M(u)}$. According to Definition 3, the periodic FDFA $\mathcal{F}$ of L captures every decomposition of $uv^\omega \in \mathrm{UP}(L)$. Moreover, the language of the progress DFA A^u of $\mathcal{F}$ is exactly the set $\{v \in \Sigma^+ \mid uv^\omega \in L\}$. Take the periodic FDFA of $L = a^\omega + ab^\omega$ in Fig. 2 as an example where the leading DFA M is given in the column labeled with "Leading" and the progress DFAs are in the column labeled with "Periodic". There are three equivalence classes of the periodic right congruence $\approx_P^{aa}$, namely $[\epsilon]_{\approx_P^{aa}}$, $[a]_{\approx_P^{aa}}$ and $[b]_{\approx_P^{aa}}$. We can check that a is not in the equivalence classes $[\epsilon]_{\approx_P^{aa}}$ or $[b]_{\approx_P^{aa}}$ since there exists a finite word ϵ such that $aa(a\epsilon)^\omega \in L$ while $aa(\epsilon\epsilon)^\omega \notin L$ and $aa(b\epsilon)^\omega \notin L$. The word ϵ does not belong to the equivalence class $[b]_{\approx_P^{aa}}$ since there exists a word a such that $aa(ba)^\omega \notin L$ while $aa(\epsilon a)^\omega \in L$. There is a transition from the state a to the state b labeled with b in the progress DFA A^{aa} since the word ab belongs to the equivalence class $[b]_{\approx_P^{aa}}$. The state a is an accepting state of A^{aa} since $aa(a)^\omega \in L$ according to Definition 3. One can easily verify that the periodic FDFA indeed captures all possible decompositions of the ultimately periodic words a^ω and ab^ω in the form of (ϵ, a^+) (by A^ϵ), (a, a^+) (by A^a), (a, b^+) (by A^a), (aa^+, a^+) (by A^{aa}) and (ab^+, b^+) (by A^{ab}).

The second type of canonical FDFAs is the syntactic FDFA constructed from the FORC as defined by Maler and Staiger in [38]. The leading DFA in the syntactic FDFA is the same as that of the periodic FDFA; they are different from each other by the definitions of the progress DFAs. Assume u is a state in the leading DFA. The progress DFA A_P^u in the periodic FDFA of L accepts the regular language $\{v \in \Sigma^+ \mid uv^\omega \in L\}$. In contrast, the progress DFA A_S^u in the syntactic FDFA accepts the regular language $\{v \in \Sigma^+ \mid u \backsim_L uv \wedge uv^\omega \in L\}$. If we construct from the right congruence $\backsim_L$ the leading DFA M, we have that $M(u) = M(uv)$ for each $v \in L(A_S^u)$, as $u \backsim_L uv$ holds. In other words, the syntactic FDFA only captures the decompositions (u, v) of $uv^\omega \in L$ such that M will go back to the state $M(u)$ after reading the period v from $M(u)$. This minor change of the ultimately periodic words captured by the syntactic FDFA can make a big difference as it has been shown in [1] that there exists some ω-regular language L for which the number of states in the syntactic FDFA is exponentially smaller than that in the periodic FDFA.

Definition 4 (Syntactic FDFA [1]). Let L be an ω-regular language and the syntactic FDFA $\mathcal{F} = (M, \{A^q\})$ of L is defined as follows.
The leading DFA M is defined the same as in Definition 3.

We define the syntactic right congruence $\approx_S^u$ for each progress DFA A^u of $\mathcal{F}$ as follows:

$$x \approx_S^u y \text{ iff } ux \backsim_L uy \text{ and } \forall v \in \Sigma^*. uxv \backsim_L u \Longrightarrow (u(xv)^\omega \in L \Longleftrightarrow u(yv)^\omega \in L).$$

The progress DFA A^u of the state $[u]_{\backsim_L} \in \Sigma^*/_{\backsim_L}$ is defined similarly as in Definition 3 except that the equivalence relation $\approx_S^u$ is used for the DFA construction. The set of accepting states F_S is the set of equivalence classes $[v]_{\approx_S^u}$ for which $uv \backsim_L u \wedge uv^\omega \in L$.

An example of the syntactic FDFA for $L = a^\omega + ab^\omega$ is given in Fig. 2, which is also considered in [1]. In [1], the progress DFA A^a in the syntactic FDFA is not correct since there is a transition from the state ab to the state b via the letter a. By the definition of $\approx_S^a$ in Definition 4, aba is not in the equivalence class $[b]_{\approx_S^a}$ since $a \cdot aba \not\backsim_L a \cdot b$. Recall that if aba and b belong to the same equivalence class of $\approx_S^a$, $a \cdot aba$ and $a \cdot b$ have to be in the same equivalence class of $\backsim_L$ first. However, it is easy to see that $a \cdot aba \not\backsim_L a \cdot b$ since there exists a word b^ω that can distinguish them. We remark that the decomposition (a, a) of a^ω is captured by the periodic FDFA but not by the syntactic FDFA in Fig. 2 since the language of A^a of the syntactic FDFA is empty while the language of A^a of the periodic FDFA is not.

The syntactic FDFA constructed by Definition 4 can have redundant states for some ω-regular languages. Let us consider the progress DFAs A^ϵ and A^a of the syntactic FDFA in Fig. 2: they both accept nothing while $\approx^\epsilon_S$ and $\approx^a_S$ have 5 and 4 equivalence classes respectively. Therefore, Angluin and Fisman proposed the use of the recurrent FDFA [1]. A progress DFA A^u_R of the recurrent FDFA of L accepts the same language as the progress DFA A^u_S of the syntactic FDFA. The difference is that A^u_R is the minimal DFA recognizing the regular language $L(A^u_S)$.

Definition 5 (Recurrent FDFA [1]). Let L be an ω-regular language and the recurrent FDFA $\mathcal{F} = (M, \{A^q\})$ of L is defined as follows.
The leading DFA M is defined the same as in Definition 3.

We define the recurrent right congruence $\approx^u_R$ for each progress DFA A^u of $\mathcal{F}$ as follows:

$$x \approx^u_R y \text{ iff } \forall v \in \Sigma^*.(uxv \backsim_L u \wedge u(xv)^\omega \in L) \Longleftrightarrow (uyv \backsim_L u \wedge u(yv)^\omega \in L).$$

The progress DFA A^u of the state $[u]_{\backsim_L} \in \Sigma^*/_{\backsim_L}$ is defined similarly as in Definition 4 except that we use the equivalence relation $\approx^u_R$ for the DFA construction.

Different from the syntactic FDFA which is associated to a Muller automaton [38], the recurrent right congruence $\approx^u_R$ focuses on the regular language $R^u = \{v \in \Sigma^+ \mid u \backsim_L uv \wedge uv^\omega \in L\}$ that the progress DFA A^u should accept. The definition of $\approx^u_R$ is an instantiation of the right congruence $\backsim_{R^u}$ for the regular language R^u where $x \backsim_{R^u} y$ iff $\forall v \in \Sigma^*.xv \in R^u \Longleftrightarrow yv \in R^u$ for any $x, y \in \Sigma^*$. Indeed, for any $x_1, x_2 \in \Sigma^*$, we have that $x_1 \approx^u_R x_2$ iff $x_1 \backsim_{R^u} x_2$. Therefore, we can see that the right congruences $\approx^\epsilon_R$ and $\approx^a_R$ of $L = a^\omega + ab^\omega$ in Fig. 2 both have only one equivalence class, which is the only equivalence class needed for the empty language.

As aforementioned, we learn a BA by learning an FDFA since there are no corresponding right congruences for general BAs. As you will see in Sect. 7, a smaller learned FDFA often results in a smaller learned BA. The reason why we keep both the periodic FDFA and the recurrent FDFA for the BA learning algorithm in this paper is due to the following facts stated in [1] for a fixed ω-regular language L.

- We mentioned before that the periodic FDFA can be exponentially larger than the syntactic FDFA for some ω-regular language L.
- The recurrent FDFA of L is at least not larger than the syntactic FDFA of L.
- There exists some ω-regular language L such that the corresponding recurrent FDFA is larger than the corresponding periodic FDFA.

Thus, we consider both periodic FDFAs and recurrent FDFAs in the BA learning algorithm, as the recurrent FDFA and the periodic FDFA are incomparable regarding the number of states. The reason why we keep syntactic FDFAs in the paper is that according to our experiments, learning Büchi automata via learning syntactic FDFAs performs quite well in practice. We also observe that the right congruences of a syntactic FDFA have stronger ability to discover new states in the learning procedure than its other counterparts; see Fig. 6 in Sect. 6.1.

In the following, we present Proposition 1 to show that all the three types of canonical FDFAs accept a special class of ultimately periodic words of L.

Proposition 1. *Let L be an ω-regular language and $\mathcal{F} = (M, \{A^u\})$ the corresponding periodic (syntactic, recurrent) FDFA of L. For any $u, v \in \Sigma^*$, if (u, v) is accepted by $\mathcal{F}$ then (u, v^k) is also accepted by $\mathcal{F}$ for every $k \geq 1$.*

Proof. Let $\tilde{u} = M(u)$ and $\widetilde{v^k} = A^{\tilde{u}}(v^k)$. We then have that $v^k \approx^{\tilde{u}}_K \widetilde{v^k}$ for every $k \geq 1$ where $K \in \{P, S, R\}$. This is because $\widetilde{v^k} = A^{\tilde{u}}(\widetilde{v^k}) = A^{\tilde{u}}(v^k)$ which means that v^k is also in the equivalence class $[\widetilde{v^k}]$. Our goal is to prove that (u, v^k) is also accepted by $\mathcal{F}$, i.e., $uv^k \backsim_M u$ and $\widetilde{v^k}$ is an accepting state for every $k \geq 1$. Recall that $\backsim_M$ and $\backsim_L$ are consistent if for any $x, y \in \Sigma^*$, $x \backsim_M y \Longleftrightarrow x \backsim_L y$. Note that $\backsim_M$ and $\backsim_L$ are indeed consistent for the canonical FDFAs. Thus we have that $uv \backsim_M u$, i.e., $uv \backsim_L u$ since (u, v) is accepted by $\mathcal{F}$. It immediately follows that $uv^k \backsim_L u$ for every $k \geq 1$. Hence, the remaining proof is to show that $\widetilde{v^k}$ is an accepting state for every $k \geq 1$ in the canonical FDFAs.

- Assume that $\mathcal{F}$ is the periodic FDFA of L. If (u, v) is accepted by $\mathcal{F}$, $\tilde{v} = A^{\tilde{u}}(v)$ must be an accepting state of $A^{\tilde{u}}$. It follows that $\tilde{u}(\tilde{v})^\omega \in L$ according to Definition 3. By the definition of $\approx^{\tilde{u}}_P$, we have that $\tilde{u}(v)^\omega \in L$ since $\tilde{v} \approx^{\tilde{u}}_P v$ and $\tilde{u}(\tilde{v})^\omega \in L$ hold. It follows that $\tilde{u}(v^k)^\omega \in L$ for every $k \geq 1$. Similarly, as $\tilde{u}(v^k)^\omega \in L$ and $v^k \approx^{\tilde{u}}_P \widetilde{v^k}$ hold, we have that $\tilde{u}(\widetilde{v^k})^\omega \in L$, which indicates that the state $\widetilde{v^k}$ is an accepting state in $A^{\tilde{u}}$ for every $k \geq 1$.

- According to Definition 5, for any $x, y \in \Sigma^*$, $\tilde{u}x \backsim_L \tilde{u} \wedge \tilde{u}x^\omega \in L \Longleftrightarrow \tilde{u}y \backsim_L \tilde{u} \wedge \tilde{u}y^\omega \in L$ holds if $x \approx^{\tilde{u}}_R y$. As $x \approx^{\tilde{u}}_S y$ implies $x \approx^{\tilde{u}}_R y$, we also have above result if $x \approx^{\tilde{u}}_S y$. In the following, $\approx^{\tilde{u}}_K$ can be replaced by $\approx^{\tilde{u}}_S$ and by $\approx^{\tilde{u}}_R$.

  Let $\mathcal{F}$ be the syntactic FDFA or the recurrent FDFA of L. If (u, v) is accepted by $\mathcal{F}$, we have that $\tilde{u}\tilde{v} \backsim_L \tilde{u}$ and $\tilde{u}(\tilde{v})^\omega \in L$ according to Definition 4 and Definition 5. From the fact that $v \approx^{\tilde{u}}_K \tilde{v}$, we also have that $\tilde{u}v \backsim_L \tilde{u}$ and $\tilde{u}(v)^\omega \in L$, which implies that $\tilde{u}v^k \backsim_L \tilde{u}$ and $\tilde{u}(v^k)^\omega \in L$ for every $k \geq 1$. Similarly, as $v^k \approx^{\tilde{u}}_K \widetilde{v^k}$ holds, it follows that $\tilde{u}\widetilde{v^k} \backsim_L \tilde{u}$ and $\tilde{u}(\widetilde{v^k})^\omega \in L$, which indicates that $\widetilde{v^k}$ is an accepting state in $A^{\tilde{u}}$ for every $k \geq 1$.

Hence we complete the proof. ■

Lemma 1 ([1]). *Let $\mathcal{F}$ be the periodic (syntactic, recurrent) FDFA of an ω-regular language L. Then $UP(\mathcal{F}) = UP(L)$.*

Lemma 2 ([54]). *Let $\mathcal{F}$ be the periodic (syntactic, recurrent) FDFA of an ω-regular language L. One can construct a BA recognizing L from $\mathcal{F}$.*

Lemma 3 ([37]). *Let $U, V \subseteq \Sigma^*$ be two languages such that $UV^* = U$ and $V^+ = V$. Then if $w \in UP(UV^\omega)$, there must exist two words $u \in U$ and $v \in V$ such that $w = uv^\omega$.*

4. Büchi Automata Learning Framework based on FDFAs

We begin with an introduction of the framework on learning a BA (respectively, LDBA) recognizing an unknown ω-regular language L, as depicted in Fig. 3.

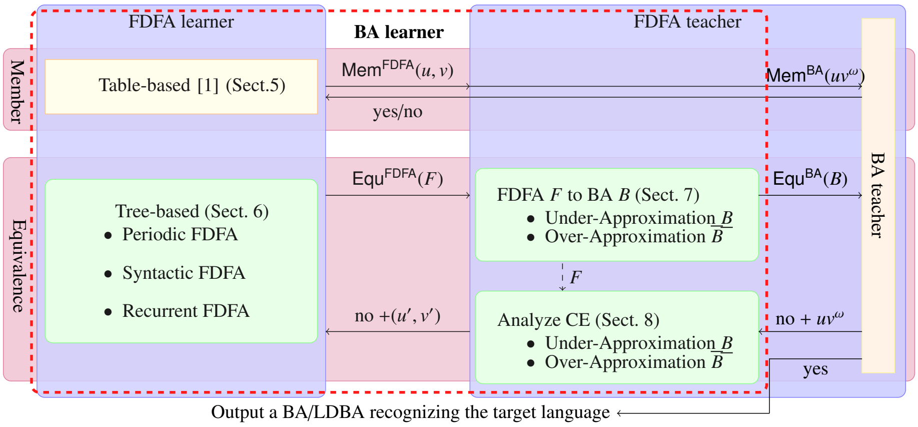

FDFA learner
BA learner
FDFA teacher
Member
Table-based [1] (Sect.5)
$\mathsf{Mem}^{\mathsf{FDFA}}(u,v)$
yes/no
$\mathsf{Mem}^{\mathsf{BA}}(uv^\omega)$
BA teacher
Equivalence
$\mathsf{Equ}^{\mathsf{FDFA}}(F)$
Tree-based (Sect. 6)
• Periodic FDFA
• Syntactic FDFA
• Recurrent FDFA
FDFA F to BA B (Sect. 7)
• Under-Approximation $\underline{B}$
• Over-Approximation $\overline{B}$
$\mathsf{Equ}^{\mathsf{BA}}(B)$
F
no $+(u',v')$
Analyze CE (Sect. 8)
• Under-Approximation $\underline{B}$
• Over-Approximation $\overline{B}$
no + uv^ω
yes
Output a BA/LDBA recognizing the target language

Figure 3: Overview of the learning framework based on FDFA learning. The components in 🟨 boxes are the results from the existing works. The components in 🟩 boxes are our new contributions.

Overview of the framework. First, we assume that we already have a BA teacher who knows the unknown ω-regular language L and can answer *membership* and *equivalence* queries about L. More precisely, a membership query $\mathsf{Mem}^{\mathsf{BA}}(uv^\omega)$ asks the teacher if $uv^\omega \in L$. While an equivalence query $\mathsf{Equ}^{\mathsf{BA}}(B)$ asks whether a BA B accepts L. The BA teacher will answer "yes" when $L(B) = L$, otherwise it will return "no" as well as a counterexample $uv^\omega \in L \ominus L(B)$, which is possible due to Theorem 1.

The BA learner, shown in Fig. 3 surrounded by the thick dashed rounded box, is built based on an FDFA learner. Note that one can place any FDFA learning algorithm to the FDFA learner component. For instance, one can use the FDFA learner from [1] which employs tables to store query results, or an FDFA learner using classification trees proposed in this paper. In order to learn a BA from the BA teacher, the BA learner first uses the FDFA learner to learn an FDFA by means of membership and equivalence queries about the target FDFA. This raises a problem that the BA learner has to solve: the BA learner needs an FDFA teacher to answer membership and equivalence queries about the target FDFA of L yet there is only a BA teacher available to answer queries about the target language L. To solve this problem, the FDFA teacher has been designed based on a BA teacher, a transformation component that transforms an FDFA F to a BA B and a counterexample analysis component. In the following, we give an overview of the FDFA learner and the FDFA teacher used in this framework.

The FDFA learner: The FDFA learner component will be introduced in Sect. 5 and Sect. 6. We will first briefly review the table-based FDFA learning algorithms [1] in Sect. 5. Then our tree-based learning algorithm for three types of canonical FDFAs will be introduced in Sect. 6. The algorithm is inspired by the tree-based L^* learning algorithm [3] and the TTT learning algorithm in [41]. Nevertheless, applying the tree structures to learn the FDFAs is not a trivial task. For example, instead of a binary tree used in [3], we need to use K-ary trees to learn syntactic FDFAs. The use of K-ary trees complicates the procedure of refining the classification trees and the automata construction. Besides that a node of a K-ary tree can have more than two children, the difference between a learning algorithm with a binary tree and that with a K-ary tree is that we may discover some new leaf nodes, i.e., new equivalence classes, when constructing a DFA from a K-ary tree. More details will be provided in Sect. 6.

The FDFA teacher: The task of the FDFA teacher is to answer queries $\mathsf{Mem}^{\mathsf{FDFA}}(u, v)$ and $\mathsf{Equ}^{\mathsf{FDFA}}(F)$ posed by the FDFA learner. Answering $\mathsf{Mem}^{\mathsf{FDFA}}(u, v)$ is easy. The FDFA teacher just needs to redirect the result of $\mathsf{Mem}^{\mathsf{BA}}(uv^{\omega})$ to the FDFA learner. Answering equivalence query $\mathsf{Equ}^{\mathsf{FDFA}}(F)$ is more tricky.

From FDFAs to BAs. The FDFA teacher needs to transform an FDFA F to a BA B to pose an equivalence query $\mathsf{Equ}^{\mathsf{BA}}(B)$. In Sect. 7, we show that, in general, it is impossible to build a BA B from an FDFA F such that $\mathrm{UP}(L(B)) = \mathrm{UP}(F)$, as there exists an FDFA $\mathcal{F}$ accepting a non-regular ω-language. In addition, such kind of FDFAs $\mathcal{F}$ can be learned as an intermediate conjectured FDFA F during the BA learning procedure. Therefore in Sect. 7, we propose two methods to approximate $\mathrm{UP}(F)$, namely the *under-approximation* and the *over-approximation* methods. As the name indicates, the under-approximation (respectively, over-approximation) method constructs a BA B from F such that $\mathrm{UP}(L(B)) \subseteq \mathrm{UP}(F)$ (respectively, $\mathrm{UP}(F) \subseteq \mathrm{UP}(L(B))$). Unless stated otherwise, only one of the two approximation methods will be used to perform the transformation from FDFAs to BAs in the whole BA learning procedure.

The under-approximation method is modified from the algorithm in [37]. Note that if the FDFA F belongs to one type of canonical FDFAs, the BA B built by the under-approximation method recognizes exactly $\mathrm{UP}(F)$, i.e., $\mathrm{UP}(L(B)) = \mathrm{UP}(F)$, which makes it a complete method for the BA learning (Lemmas 1 and 2). It follows that, in the worst case, the BA learning algorithm with the under-approximation method has to first learn a canonical FDFA F of L for constructing the right conjectured BA B such that $\mathrm{UP}(L(B)) = \mathrm{UP}(F)$.

As for the over-approximation method, we only guarantee to get a BA B such that $\mathrm{UP}(L(B)) = \mathrm{UP}(F)$ if F is a special kind of canonical FDFAs, which thus makes our learning algorithm with the over-approximation method an incomplete algorithm. The BA learning algorithm with the over-approximation method may terminate with an error when the counterexample analysis component fails to give a valid counterexample to the FDFA learner, which will be detailed in Sect. 8. Nevertheless, in the worst case, the over-approximation method produces a BA whose number of states is only quadratic in the size of the FDFA. In contrast, the number of states in the BA constructed by the under-approximation method is cubic in the size of the FDFA in the worst case. Therefore, we also consider the over-approximation method in the paper.

Counterexample Analysis. If the FDFA teacher receives "no" and a counterexample uv^{ω} from the BA teacher, the FDFA teacher has to return "no" and a valid decomposition (u', v') that can be used by the FDFA learner to refine F. In Sect. 8, we show how the FDFA teacher chooses a pair (u', v') from uv^{ω} that allows the FDFA learner to refine the current FDFA F. As the dashed line with a label F in Fig. 3 indicates, we need the current conjectured FDFA F to analyze the counterexample. The under-approximation and the over-approximation methods of FDFA to BA translation require different counterexample analysis procedures. More details will be provided in Sect. 8.

Once the BA teacher answers "yes" for the equivalence query $\mathsf{Equ}^{\mathsf{BA}}(B)$, the FDFA teacher will terminate the learning procedure and output a BA recognizing L. We remark that the output BA can be a nondeterministic BA or a limit deterministic BA.

5. Table-based Learning Algorithm for FDFAs

In this section, we briefly introduce the table-based FDFA learner in [1] under the assumption that we have an FDFA teacher who knows the target FDFA. It employs a structure called *observation table* [4] to organize the results obtained from queries and to propose candidate FDFAs. The table-based FDFA learner simultaneously runs several instances of DFA learners. The DFA learners are very similar to the L^* algorithm [4], except that they use different conditions to decide if two strings belong to the same state (based on Definitions 3, 4 and 5). More precisely, the FDFA learner uses one DFA learner L^*_M for the leading DFA M, and for each state u in M, one DFA learner $L^*_{A^u}$ for each progress DFA A^u. The table-based learning procedure works as follows. The learner L^*_M first closes the observation table by posing membership queries and then constructs a candidate for the leading DFA M. For every state u in M, the table-based algorithm runs an instance of the DFA learner $L^*_{A^u}$ to infer the progress DFA A^u. When all DFA learners propose the candidate DFAs, the FDFA learner assembles them to an FDFA $\mathcal{F} = (M, \{A^u\})$ and then poses an equivalence query for it. The FDFA teacher will either return *"yes"* which means the learning algorithm succeeds or return *"no"* accompanied by a counterexample. Once receiving the counterexample, the table-based algorithm will decide which DFA learner should refine its candidate DFA. We refer interested readers to [1] for more details on the table-based algorithm.

6. Tree-based Learning Algorithm for FDFAs

In this section, we provide our tree-based learning algorithm for the FDFAs under the assumption that we have an FDFA teacher knowing the target FDFA. To that end, we first define the classification tree structure for the FDFA learning in Sect. 6.1 and then present the tree-based learning algorithm in Sect. 6.2.

6.1. Classification Tree Structure in Learning

We first present our classification tree structure for the FDFA learning. Compared to the classification tree defined in [3, 41], ours is not restricted to be a binary tree. Formally, a classification tree is a tuple $\mathcal{T} = (N, r, L_n, L_e)$ where $N = I \cup T$ is a set of nodes consisting of the set I of *internal nodes* and the set T of *terminal nodes*, the node $r \in N$ is the root of the tree and $L_n : N \to \Sigma^* \cup (\Sigma^* \times \Sigma^*)$ labels an internal node with an *experiment* and a terminal node with a *state*. An experiment $e \in \Sigma^* \cup (\Sigma^* \times \Sigma^*)$ is a finite word or an ω-word used to distinguish words from different equivalence classes of the right congruence for the target automaton; while a state $u \in \Sigma^*$ is a finite representative word of an equivalence class of the right congruence.

Intuitively, on learning a target automaton, a state $u \in \Sigma^*$ is the representative of a unique state in the target automaton we have discovered so far. If there are two words $u, u' \in \Sigma^*$ such that $u \neq u'$ and they are representatives of different states in the target automaton, then we can always find an experiment e to distinguish u and u' according to the right congruence of the target automaton. For instance, given the periodic FDFA of $L = a^\omega + ab^\omega$ as depicted in Fig. 2, finite words a and b are two different states in the classification tree for the leading DFA M. More precisely, a and b can be distinguished by an experiment (a, a) since $a \cdot aa^\omega \in L$ while $b \cdot aa^\omega \notin L$. We notice that in the classification tree for M, an experiment is an ultimately periodic word w represented by a decomposition of w, while the experiments in the classification trees for progress DFAs are finite words. The function $L_e : I \times D \to N$ maps a parent node and a label to its corresponding child node; the set of labels D will be specified below.

During the learning procedure, we maintain a *leading tree* $\mathcal{T}$ for M, and for every state u in M, a *progress tree* $\mathcal{T}_u$ for the progress DFA A^u. For every classification tree, we define a tree experiment function $\mathbf{TE} : \Sigma^* \times (\Sigma^* \cup (\Sigma^* \times \Sigma^*)) \to D$. Intuitively, $\mathbf{TE}(x, e)$ computes the entry value at row (state) x and column (experiment) e of an observation table in the table-based learning algorithms and it also takes all possible inputs from $\Sigma^* \times (\Sigma^* \cup (\Sigma^* \times \Sigma^*))$. As the experiments for the leading tree and the progress trees are different, we actually have $\mathbf{TE} : \Sigma^* \times (\Sigma^* \times \Sigma^*) \rightarrow D$ for the leading tree and $\mathbf{TE} : \Sigma^* \times \Sigma^* \rightarrow D$ for progress trees. The labels of the nodes in a classification tree $\mathcal{T}$ satisfy the following invariants. Let $t \in T$ be a terminal node labeled with a state $x = L_n(t)$. Let $t' \in I$ be an ancestor node of t labeled with an experiment $e = L_n(t')$. Then the child of t' following the label $\mathbf{TE}(x, e)$, i.e., $L_e(t', \mathbf{TE}(x, e))$, is either the node t or an ancestor node of t. Figure 4 depicts a leading tree $\mathcal{T}$ of M in Fig. 2 for $L = a^\omega + ab^\omega$. The dashed line is for the F label and the solid one is for the T label. The tree experiment function $\mathbf{TE} : \Sigma^* \times (\Sigma^* \times \Sigma^*) \rightarrow \{\mathsf{F}, \mathsf{T}\}$ is defined as $\mathbf{TE}(u, (x, y)) = \mathsf{T}$ iff $uxy^\omega \in L$ for any $u, x, y \in \Sigma^*$. There are 4 internal nodes, namely i_1, i_2, i_3 and i_4, and 5 terminal nodes, namely t_1, t_2, t_3, t_4, and t_5. One can check that all nodes of $\mathcal{T}$ indeed satisfy aforementioned invariants. For instance, let $t = t_2$ be the terminal node and we have $x = L_n(t) = ab$. t_2 has two ancestors, namely i_1 and i_2. Let $t' = i_1$ and we have $e = L_n(t') = (\epsilon, a)$. The child of t' following the label $\mathbf{TE}(ab, (\epsilon, a)) = \mathsf{F}$ is i_2, which is an ancestor of t_2.

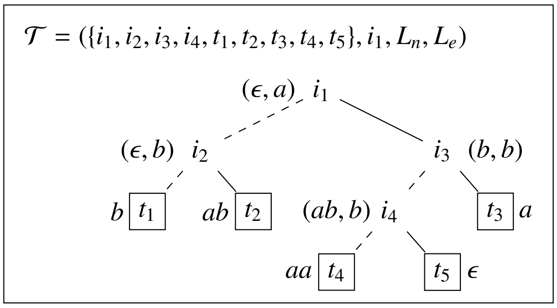


Figure 4: An example of the leading tree $\mathcal{T}$ for $L = a^\omega + ab^\omega$.

The leading tree $\mathcal{T}$: The leading tree $\mathcal{T}$ for M is a binary tree with labels $D = \{\mathsf{F}, \mathsf{T}\}$. We have $\mathbf{TE}(u, (x, y)) = \mathsf{T}$ iff $uxy^\omega \in L$ (recall the definition of $\backsim_L$ in Sect. 2) where $u, x, y \in \Sigma^*$. Intuitively, each internal node n in $\mathcal{T}$ is labeled by an experiment xy^ω represented as (x, y). For any word $u \in \Sigma^*$, $uxy^\omega \in L$ (or $uxy^\omega \notin L$) implies that the equivalence class of u lies in the T-subtree (or F-subtree) of n. One example of the leading tree $\mathcal{T}$ for M of Fig. 2 is depicted in Fig. 4. One can see that every label of the terminal nodes corresponds to a state of M.

The progress tree $\mathcal{T}_u$: The progress tree $\mathcal{T}_u$ of the state u and the corresponding function $\mathbf{TE}(x, e)$ are defined based on the right congruences $\approx^u_P$, $\approx^u_S$, and $\approx^u_R$ of the canonical FDFAs introduced in Definitions 3, 4 and 5.

Periodic FDFAs. A progress tree for the periodic FDFA is also a binary tree labeled with $D = \{\mathsf{F}, \mathsf{T}\}$. We have $\mathbf{TE}(x, e) = \mathsf{T}$ iff $u(xe)^\omega \in L$ where $x, e \in \Sigma^*$. Intuitively, for any $u, u' \in \Sigma^*$ such that $u \not\approx^u_P u'$, there must exist some experiment $e \in \Sigma^*$ such that $\mathbf{TE}(u, e) \neq \mathbf{TE}(u', e)$. For instance, the progress tree $\mathcal{T}_{aa}$ for the progress DFA A^{aa} of the periodic FDFA of Fig. 2 is depicted in Fig. 5. We can see that $\mathbf{TE}(\epsilon, a) = \mathsf{T}$ since $aa(\epsilon a)^\omega \in L$ while $\mathbf{TE}(b, a) = \mathsf{F}$ since $aa(ba)^\omega \notin L$. Thus in $\mathcal{T}_{aa}$, the experiment a of the internal node i_2 can distinguish the states ϵ and b.

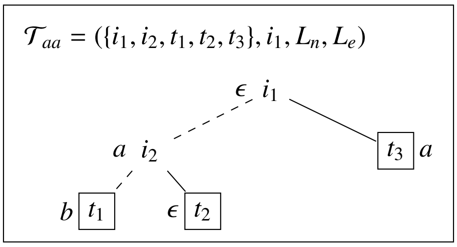

$\mathcal{T}_{aa} = (\{i_1, i_2, t_1, t_2, t_3\}, i_1, L_n, L_e)$
ϵ i_1
a i_2
t_3 a
b t_1
ϵ t_2

Figure 5: An example of the progress tree $\mathcal{T}_{aa}$ for $L = a^\omega + ab^\omega$.

Syntactic FDFAs. A progress tree for the syntactic FDFA is a K-ary tree labeled with $D = Q \times \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$ where Q is the set of states in the current leading DFA M and $K = 3|Q|$. Note that when the current leading tree $\mathcal{T}$ is fixed, one can immediately construct M according to Definition 6, which will be detailed in Sect. 6.2. Therefore, we can fix a leading DFA M in the definition of **TE** function. For all $x, e \in \Sigma^*$, we have $\mathbf{TE}(x, e) = (M(ux), t)$, where $t = \mathsf{A}$ iff $u = M(uxe) \wedge u(xe)^\omega \in L$, $t = \mathsf{B}$ iff $u = M(uxe) \wedge u(xe)^\omega \notin L$, and $t = \mathsf{C}$ iff $u \neq M(uxe)$.

Consider the progress tree $\mathcal{T}_{aa}$ for A^{aa} of the syntactic FDFA of Fig. 2 depicted in Fig. 6: the dashed line, the dotted line and the solid line are labeled by L_e with $\mathbf{TE}(a, \epsilon) = (M(aaa), A) = (aa, A)$, $\mathbf{TE}(\epsilon, \epsilon) = (M(aa\epsilon), B) = (aa, B)$ and $\mathbf{TE}(b, \epsilon) = (M(aab), C) = (b, C)$ respectively. Therefore for $\mathcal{T}_{aa}$, only the experiment ϵ of the internal node i_1 is needed to distinguish the states ϵ, a and b from each other while for the periodic and the recurrent progress trees at least two experiments are needed. It indicates that the syntactic right congruences may identify more states with the same amount of experiments compared to the two others.

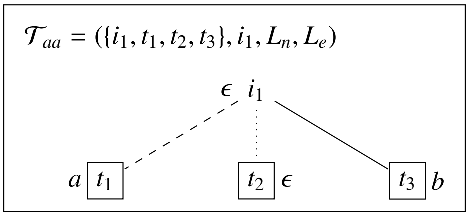

$\mathcal{T}_{aa} = (\{i_1, t_1, t_2, t_3\}, i_1, L_n, L_e)$
ϵ i_1
a t_1
t_2 ϵ
t_3 b

Figure 6: The progress tree $\mathcal{T}_{aa}$ for the syntactic FDFA of $L = a^\omega + ab^\omega$.

Recurrent FDFAs. A progress tree for the recurrent FDFA is a binary tree labeled with $D = \{\mathsf{F}, \mathsf{T}\}$. We have $\mathbf{TE}(x, e) = \mathsf{T}$ iff $u(xe)^\omega \in L \wedge u = M(uxe)$ where $x, e \in \Sigma^*$. The progress tree $\mathcal{T}_{aa}$ for A^{aa} of the recurrent FDFA of Fig. 2 is also the one depicted in Fig. 5.

6.2. *Tree-based Learning Algorithm*

The tree-based learning algorithm first initializes the leading tree $\mathcal{T}$ and the progress tree $\mathcal{T}_\epsilon$ as a tree with only one terminal node r labeled by ϵ.

Definition 6. From a classification tree $\mathcal{T} = (N, r, L_n, L_e)$, the learner constructs a candidate of the leading DFA $M = (\Sigma, Q, \epsilon, \delta)$ or a progress DFA $A^u = (\Sigma, Q, \epsilon, \delta, F)$ as follows. The set of states is $Q = \{L_n(t) \mid t \in T\}$. For $s \in Q$ and $a \in \Sigma$, the transition function $\delta(s, a)$ is constructed by the following procedure. Initially the current node $n := r$. If n is a terminal node, it returns $\delta(s, a) = L_n(n)$. Otherwise, it picks a unique child n' of n with $L_e(n, \mathbf{TE}(sa, L_n(n))) = n'$, updates the current node to n', and repeats the procedure [2]. By Definitions 3, 4 and 5, the set of accepting states F of a progress DFA A^u can be identified from the structure of M with the help of membership queries, where u is a state of M. For the periodic FDFA, $F = \{v \mid uv^\omega \in L, v \in Q\}$ and for the syntactic and the recurrent FDFAs, $F = \{v \mid uv \backsim_M u, uv^\omega \in L, v \in Q\}$ where Q is the set of states of A^u.

Figure 7 depicts the periodic progress tree $\mathcal{T}_a$ and its corresponding progress DFA A^a for $L = a^\omega + ab^\omega$. The dashed line is for the F label while the solid one is for the T label. The tree experiment function is defined as $\mathbf{TE}(x, y) = \mathsf{T}$ iff $a(xy)^\omega \in L$. To construct $A^a = (\Sigma, Q, \epsilon, \delta, F)$ from $\mathcal{T}_a$, one first has to construct the state set $Q = \{\epsilon, a, b, ab\}$ by collecting all terminal labels in $\mathcal{T}_a$. As for the transition function, we give an example to further illustrate it. For instance, $\delta(b, a)$ is decided by classifying the word ba to one of the terminal nodes in $\mathcal{T}_a$. Starting with the root i_1, we have $\mathbf{TE}(ba, L_n(i_1)) = \mathbf{TE}(ba, \epsilon) = \mathsf{F}$ since $a \cdot (ba \cdot \epsilon)^\omega \notin L$. Therefore, we go to the F-child of i_1, namely i_2. Since i_2 is not a terminal node and we have $\mathbf{TE}(ba, L_n(i_2)) = \mathbf{TE}(ba, b) = \mathsf{F}$, we go further to its F-child and finally reach the terminal node t_1 labeled with ab. Thus, we conclude that $\delta(b, a) = ab$. We identify every state $v \in Q$ such that $av^\omega \in L$ as an accepting state of A^a according to Definition 6. Therefore, we have $F = \{a, b\}$ since $a \cdot (a)^\omega \in L$ and $a \cdot (b)^\omega \in L$.

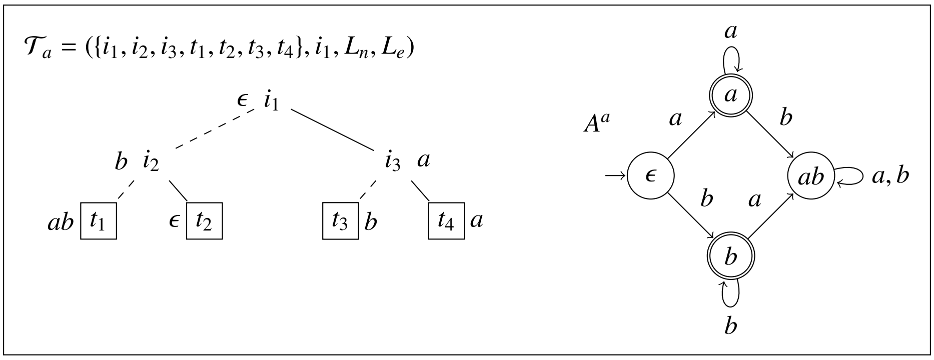


Figure 7: An example of the periodic progress tree $\mathcal{T}_a$ and the periodic progress DFA A^a for $L = a^\omega + ab^\omega$.

Whenever the learner has constructed an FDFA $\mathcal{F} = (M, \{A^u\})$, it will pose an equivalence query for $\mathcal{F}$. If the teacher returns "no" and a counterexample (u, v), the learner has to refine a classification tree and then proposes another candidate FDFA.

Definition 7 (Counterexample for the FDFA Learner). Let L be the target language and $\mathcal{F}$ the conjectured FDFA. We say that the counterexample (u, v) is

- *positive* if $uv \backsim_M u$, $uv^\omega \in \mathrm{UP}(L)$, and (u, v) is not accepted by $\mathcal{F}$, or
- *negative* if $uv \backsim_M u$, $uv^\omega \notin \mathrm{UP}(L)$, and (u, v) is accepted by $\mathcal{F}$.

[2] For the syntactic FDFA, it can happen that $\delta(s, a)$ goes to a "new" terminal node when dealing with the progress trees. A new state for the progress DFA is identified in such a case.

We remark that in our case every counterexample (u, v) from the FDFA teacher satisfies the constraint $uv \backsim_M u$, which corresponds to the *normalized factorization* form with respect to M in [1]. A normalized factorization of (u, v) with respect to M is the decomposition (x, y) such that $x = uv^i, y = v^j$ and $0 \leq i < j$ are the smallest for which $uv^i \backsim_M uv^{i+j}$ according to [1]. As $\backsim_M$ is of finite index, there must exist such i and j [1]. Let (u, v) be a counterexample for the FDFA learner defined in Definition 7; the normalized factorization (x, y) can be easily obtained by setting $x = u$ and $y = v$ since $uv \backsim_M u$. For the FDFA learning algorithm presented in [1], the normalized factorization (x, y) of the returned counterexample (u, v) is first computed and later used in the refinement of the conjectured FDFA (see Algorithm 1 in [1]). This is due to the fact that a decomposition (x, y) of a word $uv^\omega \in L$ is accepted by a canonical FDFA of L if $xy \backsim_M x$ according to Definition 2. Therefore, in order to remove (respectively, add) the rejected (respectively, accepted) decomposition in the current conjectured FDFA, the normalized factorization has to be computed first. One can check that our returned counterexample for the FDFA learner constructed in Sect. 8 respects Definition 7. The way we analyze the returned counterexamples is similar to the one applied in the table-based FDFA learning algorithm [1] so we also follow their way to present the counterexample analysis for the FDFA learner in the following.

Counterexample guided refinement of $\mathcal{F}$: Below we show how to refine classification trees with a negative counterexample (u, v). The case of a positive counterexample is symmetric. By definition, we have $uv \sim_M u$, $uv^\omega \notin \mathrm{UP}(L)$ and (u, v) is accepted by $\mathcal{F}$. Let $\tilde{u} = M(u)$: if $\tilde{u}v^\omega \in \mathrm{UP}(L)$, the refinement of the leading tree is performed, otherwise $\tilde{u}v^\omega \notin \mathrm{UP}(L)$, the progress tree $\mathcal{T}_{\tilde{u}}$ must be refined.

Refinement for the leading tree: In the leading DFA M of the conjectured FDFA, if a state p has a transition to a state q via a letter a, i.e, $q = M(pa)$, then pa has been assigned to the terminal node labeled by q during the construction of M. If one finds an experiment e such that $\mathbf{TE}(q, e) \neq \mathbf{TE}(pa, e)$, then we know that q and pa can not belong to the same state in the leading DFA. W.l.o.g., we assume $\mathbf{TE}(q, e) = \mathsf{F}$. In such a case, the leading tree can be refined by replacing the terminal node labeled with q by a tree such that (i) its root is labeled by e, (ii) its left child is a terminal node labeled by q, and (iii) its right child is a terminal node labeled by pa.

Below we discuss how to extract the required states p, q and experiment e for refining the leading tree once receiving a negative counterexample (u, v). Let $|u| = n$ and for $i \in [1 \cdots n]$, let $s_i = M(u[1 \cdots i])$ be the state reached after reading the first i-letters of u. Recall that s_i is the representative word of the state $M(u[1 \cdots i])$. In particular, $s_0 = M(\epsilon) = \epsilon$ and $s_n = M(u) = \tilde{u}$. Therefore, we can compute a sequence of results $\mathbf{TE}(s_0, (u[1 \cdots n], v)), \mathbf{TE}(s_1, (u[2 \cdots n], v))$, $\mathbf{TE}(s_2, (u[3 \cdots n], v))$ and so on, up to $\mathbf{TE}(s_n, (u[n + 1 \cdots n], v)) = \mathbf{TE}(\tilde{u}, (\epsilon, v))$. Recall that we have $w[j \cdots k] = \epsilon$ when $j > k$ as defined in Sect. 2. This sequence has different results for the first and the last experiment function since $\mathbf{TE}(s_0, (u[1 \cdots n], v)) = \mathsf{F}$ while $\mathbf{TE}(\tilde{u}, (\epsilon, v)) = \mathsf{T}$ by the assumption that $uv^\omega \notin \mathrm{UP}(L)$ and $\tilde{u}v^\omega \in \mathrm{UP}(L)$.

Therefore, there must exist the smallest $j \in [1 \cdots n]$ such that $\mathbf{TE}(s_{j-1}u[j], (u[j+1 \cdots n], v)) \neq \mathbf{TE}(s_j, (u[j + 1 \cdots n], v))$. It follows that we can use the experiment $e = (u[j + 1 \cdots n], v)$ to distinguish $q = s_j$ and $pa = s_{j-1}u[j]$.

Example 1. *Consider the intermediate conjectured FDFA $\mathcal{F}$ in Fig. 1 during the process of learning $L = a^\omega + b^\omega$. The corresponding leading tree $\mathcal{T}$ and the progress tree $\mathcal{T}_\epsilon$ are depicted on the left of Fig. 8. The dashed line is for the* F *label and the solid one is for the* T *label. Suppose the FDFA teacher returns a negative counterexample (ab, b). The leading tree must be refined since $M(ab)b^\omega = b^\omega \in L$. We find an experiment (b, b) to distinguish ϵ and a using the procedure*

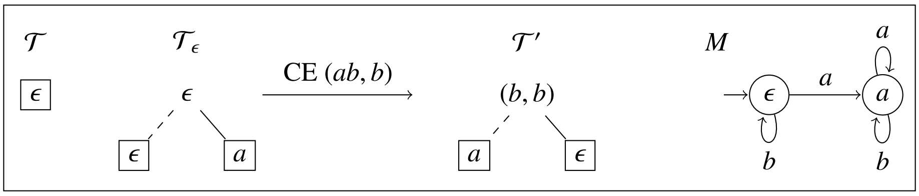


Figure 8: Refinement of the leading tree and the corresponding leading DFA

above and update the leading tree $\mathcal{T}$ to $\mathcal{T}'$. The leading DFA M constructed from $\mathcal{T}'$ is shown on the right of Fig. 8.

Refinement for the progress tree: Recall that $\tilde{u} \cdot v^\omega \notin \mathrm{UP}(L)$ and thus the algorithm is to refine the progress tree $\mathcal{T}_{\tilde{u}}$. Let $|v| = n$ and for $i \in [1 \cdots n]$, let $s_i = A^{\tilde{u}}(v[1 \cdots i])$ be the state reached after reading $v[1 \cdots i]$. In particular, $s_0 = A^{\tilde{u}}(\epsilon) = \epsilon$ and $s_n = A^{\tilde{u}}(v) = \tilde{v}$. Similarly, we have a sequence of results $\mathbf{TE}(s_0, v[1 \cdots n]), \mathbf{TE}(s_1, v[2 \cdots n])$ and so on, up to $\mathbf{TE}(s_n, v[n + 1 \cdots n]) = \mathbf{TE}(\tilde{v}, \epsilon)$. This sequence has different results for the first and the last experiment function, i.e., $\mathbf{TE}(s_0, v[1 \cdots n]) \neq \mathbf{TE}(\tilde{v}, \epsilon)$, which will be explained later. Therefore, there must exist the smallest $j \in [1 \cdots n]$ such that $\mathbf{TE}(s_{j-1}v[j], v[j + 1 \cdots n]) \neq \mathbf{TE}(s_j, v[j + 1 \cdots n])$. It follows that we can use the experiment $e = v[j + 1 \cdots n]$ to distinguish $q = s_j$, $pa = s_{j-1}v[j]$ and then refine the progress tree $\mathcal{T}_{\tilde{u}}$ as follows. We refine $\mathcal{T}_{\tilde{u}}$ by replacing the terminal node labeled with s_j by a tree such that (i) its root is labeled by $e = v[j + 1 \cdots n]$, (ii) its $\mathbf{TE}(s_j, v[j + 1 \cdots n])$-subtree is a terminal node labeled by s_j, and (iii) its $\mathbf{TE}(s_{j-1}v[j], v[j + 1 \cdots n])$-subtree is a terminal node labeled by $s_{j-1}v[j]$.

To complete the refinement process, we show why $\mathbf{TE}(s_0, v) \neq \mathbf{TE}(\tilde{v}, \epsilon)$ holds.

Periodic FDFAs. As $\tilde{u}(\epsilon \cdot v)^\omega \notin \mathrm{UP}(L)$, we have $\mathbf{TE}(\epsilon, v) = \mathsf{F}$. While $\mathbf{TE}(\tilde{v}, \epsilon) = \mathsf{T}$ because $\tilde{v}$ is an accepting state and thus $\tilde{u}(\tilde{v} \cdot \epsilon)^\omega \in \mathrm{UP}(L)$. Thus $\mathbf{TE}(s_0, v) \neq \mathbf{TE}(\tilde{v}, \epsilon)$.

Syntactic FDFAs. We have $uv \backsim_M u$, i.e., $M(uv) = M(u)$ according to Definition 7. Let $\tilde{u} = M(u)$. Recall that $\tilde{u}$ is the representative word of the state $M(u)$, which indicates that $\tilde{u} = M(u) = M(\tilde{u})$. It follows that $\tilde{u} = M(\tilde{u}v)$ since $M(uv) = M(u)$. Therefore, we have $\mathbf{TE}(\epsilon, v) = (M(\tilde{u} \cdot \epsilon \cdot v), \mathrm{B}) = (\tilde{u}, \mathrm{B})$, where B is obtained here since $\tilde{u} = M(\tilde{u} \cdot \epsilon \cdot v)$ and $\tilde{u}(\epsilon \cdot v)^\omega \notin \mathrm{UP}(L)$ according to the definition of **TE** for syntactic FDFAs. Moreover, $\tilde{v}$ is an accepting state of current syntactic FDFA. It follows that $\tilde{u} = M(\tilde{u}\tilde{v})$ and $\tilde{u}(\tilde{v})^\omega \in L$ according to Definition 4. Therefore, we have $\mathbf{TE}(\tilde{v}, \epsilon) = (M(\tilde{u}\tilde{v}), \mathrm{A}) = (\tilde{u}, \mathrm{A})$ where A is obtained since $\tilde{u} = M(\tilde{u} \cdot \tilde{v} \cdot \epsilon)$ and $\tilde{u}(\tilde{v} \cdot \epsilon)^\omega \in \mathrm{UP}(L)$. Thus $\mathbf{TE}(s_0, v) \neq \mathbf{TE}(\tilde{v}, \epsilon)$.

Recurrent FDFA. Similar to the case for syntactic FDFAs, we have $\mathbf{TE}(\epsilon, v) = \mathsf{F}$ and $\mathbf{TE}(\tilde{v}, \epsilon) = \mathsf{T}$. Thus $\mathbf{TE}(s_0, v) \neq \mathbf{TE}(\tilde{v}, \epsilon)$.

Optimization: Example 1 also illustrates the fact that the counterexample (ab, b) may not be eliminated right away after the refinement. In this case, it is still a valid counterexample (assuming that the progress tree $\mathcal{T}_\epsilon$ remains unchanged). Thus one can repeatedly use the counterexample until it is eliminated as an optimization to reduce interactions with the teacher.

We introduce an immediate result of the counterexample guided refinement for $\mathcal{F}$ as Lemma 4. It shows that the tree-based learning algorithm will make progress upon receiving a counterexample, which is vital for the termination of the learning algorithm.

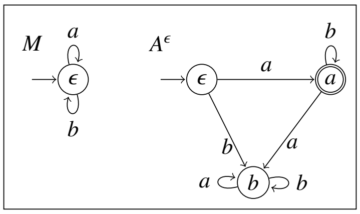


Figure 9: An FDFA $\mathcal{F}$ such that UP($\mathcal{F}$) does not characterize an ω-regular language

Lemma 4. *During the learning procedure, if the tree-based FDFA learner receives a counterexample (u, v), then there will be at least a new state added to the leading DFA M or the progress DFA $A^{\tilde{u}}$ where $\tilde{u} = M(u)$.*

7. From FDFAs to Büchi Automata

Since the FDFA teacher exploits the BA teacher for answering equivalence queries, the conversion from the conjectured FDFA into a BA is needed. Unfortunately, with the following example, we show that in general it is impossible to construct a *precise* BA B for an FDFA $\mathcal{F}$ such that UP($L(B)$) = UP($\mathcal{F}$). Note that this result has been discussed and proved for the first time in the previous version [44] of this paper.

Example 2. *Consider a non-canonical FDFA $\mathcal{F}$ in Fig. 9, we have UP($\mathcal{F}$) = $\bigcup_{n=0}^{\infty}\{a, b\}^* \cdot (ab^n)^{\omega}$. We assume that UP($\mathcal{F}$) characterizes an ω-regular language L. It follows that the index of each right congruence of the periodic FDFA F recognizing L with UP($\mathcal{F}$) = UP(F) is finite [1]. However, we can show that the right congruence $\approx_P^{\epsilon}$ of the periodic FDFA F of L, if exists, must be of infinite index. Observe that $ab^k \not\approx_P^{\epsilon} ab^j$ for any $k, j \geq 1$ and $k \neq j$, because $\epsilon \cdot (ab^k \cdot ab^k)^{\omega} \in$ UP($\mathcal{F}$) and $\epsilon \cdot (ab^j \cdot ab^k)^{\omega} \notin$ UP($\mathcal{F}$). It follows that $\approx_P^{\epsilon}$ is of infinite index. We conclude that UP($\mathcal{F}$) cannot characterize an ω-regular language.*

Therefore, in general, we cannot construct a BA B from an FDFA $\mathcal{F}$ such that UP($L(B)$) = UP($\mathcal{F}$). As mentioned in Sect. 4, we propose the under-approximation and the over-approximation methods to approximate UP($\mathcal{F}$). We propose a BA $\underline{B}$, which underapproximates UP($\mathcal{F}$), i.e., UP($L(\underline{B})$) ⊆ UP($\mathcal{F}$). For the under-approximation method, on receiving a counterexample from the BA teacher, the FDFA teacher can always find a valid counterexample for the FDFA learner defined in Definition 7 to refine the current FDFA. Moreover, if $\mathcal{F}$ is a canonical FDFA, the under-approximation method guarantees to construct a BA $\underline{B}$ such that UP($L(\underline{B})$) = UP($\mathcal{F}$), which makes it a complete method for the BA learning. Another proposal is to construct a BA $\overline{B}$ that overapproximates UP($\mathcal{F}$), i.e., UP($\mathcal{F}$) ⊆ UP($L(\overline{B})$). For the over-approximation method, given a canonical FDFA $\mathcal{F}$, that whether UP($L(\overline{B})$) = UP($\mathcal{F}$) is still unknown and the FDFA teacher may not be able to find a valid counterexample for the FDFA learner when dealing with counterexamples returned from the BA teacher. Nevertheless, the over-approximation method guarantees to construct a BA $\overline{B}$ such that UP($L(\overline{B})$) = UP($\mathcal{F}$) for a special class of canonical FDFAs $\mathcal{F}$. We keep the over-approximation method in the paper since the size of $\overline{B}$ is quadratic in the size of the conjectured FDFA while the size of $\underline{B}$ is cubic in the worst case according to Lemma 6.

We first give the main idea behind the two approximation methods and then give the formal definition of these two methods. Given an FDFA $\mathcal{F} = (M, \{A^u\})$ with $M = (\Sigma, Q, q_0, \delta)$ and A^u =

$(\Sigma, Q_u, s_u, \delta_u, F_u)$ for all $u \in Q$, we define $M_v^s = (\Sigma, Q, s, \delta, \{v\})$ and $(A^u)_v^s = (\Sigma, Q_u, s, \delta_u, \{v\})$, i.e., the DFA obtained from M and A^u by setting their initial state and accepting states to s and $\{v\}$, respectively. We define $N_{(u,v)} = \{v^\omega \mid uv \backsim_M u \wedge v \in L((A^u)_v^{s_u})\}$, which includes only the words $v \in L((A^u)_v^{s_u})$ such that $u = M(u) = M(uv)$. Therefore, according to Definition 2, it follows that $\mathrm{UP}(\mathcal{F}) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot N_{(u,v)}$ where $L(M_u^{q_0})$ contains the finite prefixes and $N_{(u,v)}$ contains the periodic finite words for every state pair (u, v).

We construct $\overline{B}$ and $\underline{B}$ by approximating the set $N_{(u,v)}$. For $\overline{B}$, we first define an FA $\overline{P}_{(u,v)} = (\Sigma, Q_{(u,v)}, s_{(u,v)}, \delta_{(u,v)}, \{f_{(u,v)}\}) = M_u^u \times (A^u)_v^{s_u}$ and let $\overline{N}_{(u,v)} = L(\overline{P}_{(u,v)})^\omega$. Then one can construct a BA $(\Sigma, Q_{(u,v)} \cup \{f\}, s_{(u,v)}, \delta_{(u,v)} \cup \delta_f, \{f\})$ recognizing $\overline{N}_{(u,v)}$ where f is a "fresh" state and $\delta_f = \{(f, \epsilon, s_{(u,v)}), (f_{(u,v)}, \epsilon, f)\}$. Note that ϵ transitions can be taken without consuming any letters and can be removed by standard methods in automata theory, see e.g., [48]. It is easy to see that for any $(v_1)^\omega \in N_{(u,v)}$, there exists a word $v' \in L(\overline{P}_{(u,v)})$ such that $(v_1)^\omega = (v')^\omega$ and for any $v_2 \in L(\overline{P}_{(u,v)})$, we have that $(v_2)^\omega \in N_{(u,v)}$. Therefore, a simple and natural way to overapproximate the set $N_{(u,v)}$ is to construct a BA accepting $L(\overline{P}_{(u,v)})^\omega$, i.e., $\overline{N}_{(u,v)}$. Intuitively, we overapproximate the set $N_{(u,v)}$ as $\overline{N}_{(u,v)}$ by adding $(v_1 \cdot v_2)^\omega$ into $N_{(u,v)}$ if $(v_1)^\omega \in N_{(u,v)}$ and $(v_2)^\omega \in N_{(u,v)}$ where $v_1, v_2 \in \Sigma^+$. For $\underline{B}$, we define an FA $\underline{P}_{(u,v)} = M_u^u \times (A^u)_v^{s_u} \times (A^u)_v^v$ and let $\underline{N}_{(u,v)} = L(\underline{P}_{(u,v)})^\omega$. One can construct a BA recognizing $\underline{N}_{(u,v)}$ using a similar construction to the case of $\overline{N}_{(u,v)}$. Intuitively, we underapproximate the set $N_{(u,v)}$ as $\underline{N}_{(u,v)}$ by only keeping $v^\omega \in N_{(u,v)}$ if $A^u(v) = A^u(v \cdot v)$ where $v \in \Sigma^+$. In Definition 8 we show how to construct BAs $\overline{B}$ and $\underline{B}$ s.t. $\mathrm{UP}(L(\overline{B})) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot \overline{N}_{(u,v)}$ and $\mathrm{UP}(L(\underline{B})) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot \underline{N}_{(u,v)}$.

Definition 8. Let $\mathcal{F} = (M, \{A^u\})$ be an FDFA where $M = (\Sigma, Q, q_0, \delta)$ and $A^u = (\Sigma, Q_u, s_u, \underline{\delta_u}, F_u)$ for every $u \in Q$. Let $(\Sigma, Q_{(u,v)}, s_{(u,v)}, \delta_{(u,v)}, \{f_{(u,v)}\})$ be a BA recognizing $\underline{N}_{(u,v)}$ (respectively $\overline{N}_{(u,v)}$). Then the BA $\underline{B}$ (respectively $\overline{B}$) is defined as the tuple

$$\left(\Sigma, Q \cup \bigcup_{u \in Q, v \in F_u} Q_{(u,v)}, q_0, \delta \cup \bigcup_{u \in Q, v \in F_u} \delta_{(u,v)} \cup \bigcup_{u \in Q, v \in F_u} \{(u, \epsilon, s_{(u,v)})\}, \bigcup_{u \in Q, v \in F_u} \{f_{(u,v)}\}\right).$$

Intuitively, we connect the leading DFA M to the BA recognizing $\underline{N}_{(u,v)}$ (respectively $\overline{N}_{(u,v)}$) by linking the state u of M and the initial state $s_{(u,v)}$ of the BA with an ϵ-transition for every state pair (u, v) where $v \in F_u$.

Figure 10 depicts the BAs $\overline{B}$ and $\underline{B}$ constructed from the FDFA $\mathcal{F}$ in Fig. 1. In the example, we can see that $b^\omega \in \mathrm{UP}(\mathcal{F})$ while $b^\omega \notin \mathrm{UP}(L(\underline{B}))$.

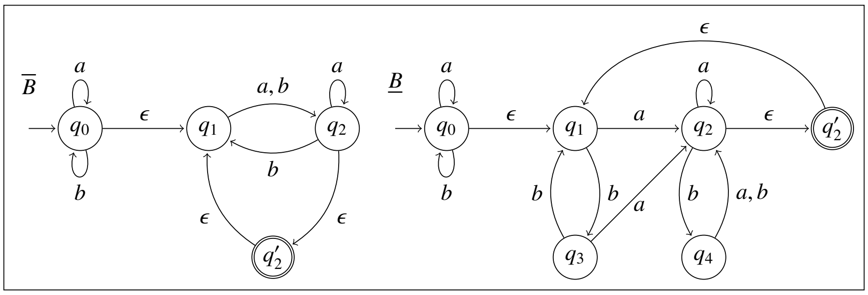


Figure 10: The NBAs $\overline{B}$ and $\underline{B}$ constructed from $\mathcal{F}$ of Fig. 1

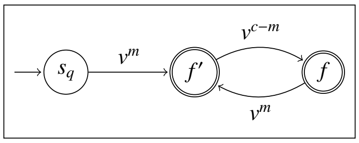


Figure 11: Illustration of $A^q(v^j) = A^q(v^{2j})$.

In the following, we introduce Lemma 5, which will be used to prove Lemma 6.

Lemma 5. *Let $\mathcal{F} = (M, \{A^u\})$ be an FDFA, and $\underline{B}$ be the BA constructed from $\mathcal{F}$ by Definition 8. If (u, v^k) is accepted by $\mathcal{F}$ for every $k \geq 1$, then $uv^\omega \in UP(L(\underline{B}))$.*

Proof. Let $\mathcal{F}$ be the FDFA where we have $M = (\Sigma, Q, q_0, \delta)$ and $A^q = (\Sigma, Q_q, s_q, \delta_q, F_q)$ for each $q \in Q$. Let $q = M(u)$ be the state in M after reading the word u. In the progress DFA A^q, we have $v^k \in L(A^q)$ for any $k \geq 1$ by assumption. Since the number of accepting states in the DFA A^q is finite, we claim that there must exist an integer, say $j \geq 1$, such that $p = A^q(v^j) = A^q(v^j \cdot v^j)$ for some $p \in F_q$, which will be proved later. Therefore, $v^j \in L(\underline{P}_{(q,p)})$ holds where $\underline{P}_{(q,p)} = M^q_q \times (A^q)^{s_q}_p \times (A^q)^p_p$. The reason why $v^j \in L(\underline{P}_{(q,p)})$ holds is that: (i) $v^j \in L(M^q_q)$ since (u, v^j) is accepted by $\mathcal{F}$ by assumption and thus $q = M(u) = M(u \cdot v^j)$; (ii) $v^j \in L((A^q)^{s_q}_p \times (A^q)^p_p)$ since $p = A^q(v^j) = A^q(v^j \cdot v^j)$ and $p \in F_q$. Recall that $\text{UP}(L(\underline{B})) = \bigcup_{q \in Q, p \in F_q} L(M^{q_0}_q) \cdot \underline{N}_{(q,p)} = \bigcup_{q \in Q, p \in F_q} L(M^{q_0}_q) \cdot (L(\underline{P}_{(q,p)}))^\omega$. Thus we have $u \cdot (v^j)^\omega = uv^\omega \in \text{UP}(L(\underline{B}))$.

Now we show that there exists an integer $j \geq 1$ such that $p = A^q(v^j) = A^q(v^j \cdot v^j)$ for some $p \in F_q$. Since A^q is a DFA with finite accepting states, there must be some $m \geq 1$ and $c > m$ such that $f' = A^q(v^m) = A^q(v^{m+c})$, which is depicted in Fig. 11. Let $j = c$. It is easy to see that the state p is actually the accepting state f in Fig. 11 since $f = A^q(v^c) = A^q(v^{c+c})$. Therefore, we complete the proof. ■

Lemma 6 (Sizes and Languages of $\underline{B}$ and $\overline{B}$). *Let $\mathcal{F}$ be an FDFA and $\underline{B}$, $\overline{B}$ be the BAs constructed from $\mathcal{F}$ by Definition 8. Let n and k be the numbers of states in the leading DFA and the largest progress DFA of $\mathcal{F}$, respectively. The numbers of states of $\underline{B}$ and $\overline{B}$ are in $O(n^2k^3)$ and $O(n^2k^2)$, respectively. Moreover, $UP(L(\underline{B})) \subseteq UP(\mathcal{F}) \subseteq UP(L(\overline{B}))$ holds and we have $UP(L(\underline{B})) = UP(\mathcal{F})$ when $\mathcal{F}$ is a canonical FDFA.*

Proof.

- Sizes of $\underline{B}$ and $\overline{B}$. In the under-approximation construction, for each state q of M, there is a progress DFA A^q of size at most k. It is easy to see that the DFA $\underline{P}_{(q,p)}$ is of size at most nk^2 for every $p \in F_q$ of A^q. Thus $\underline{B}$ is of size at most $n + nk \cdot nk^2 \in O(n^2k^3)$. The over-approximation method differs in the construction of the DFA $\overline{P}_{(u,v)}$ from the under-approximation method. It is easy to see that the DFA $\overline{P}_{(q,p)}$ is of size at most nk for every $v \in F_p$ of A^q. Therefore $\overline{B}$ is of size at most $n + nk \cdot nk \in O(n^2k^2)$.

- $\text{UP}(L(\underline{B})) \subseteq \text{UP}(\mathcal{F})$. Let w be an ultimately periodic word accepted by $\underline{B}$, i.e., $w \in \text{UP}(L(\underline{B})) = \bigcup_{q \in Q, p \in F_q} L(M^{q_0}_q) \cdot (L(\underline{P}_{(q,p)}))^\omega$. Therefore, there exist a state q of M and a state $p \in F_q$ of A^q such that $w = u \cdot v_1 \cdot v_2 \cdot v_n \cdots$ where $u \in L(M^{q_0}_q)$, $v_i \in L(\underline{P}_{(q,p)})$ for every $i \geq 1$ and q_0 is the initial state of M. According to Definition 8, $\underline{P}_{(q,p)}$ is the product of three DFAs M_q^q, $(A^q)_p^{s_q}$ and $(A^q)_p^p$ where s_q is the initial state in A^q. It follows that (i) $L(M_q^{q_0}) \cdot (L(\underline{P}_{(q,p)}))^* = L(M_q^{q_0})$ holds and (ii) $(L(\underline{P}_{(q,p)}))^+ = L(\underline{P}_{(q,p)})$ holds.

In the following we prove equation (i) and then equation (ii). First $L(M_q^{q_0}) \cdot (L(\underline{P}_{(q,p)}))^* \subseteq L(M_q^{q_0})$ holds since $L(\underline{P}_{(q,p)}) \subseteq L(M_q^q)$. Moreover, $\epsilon \in (L(\underline{P}_{(q,p)}))^*$ and it follows that $L(M_q^{q_0}) \subseteq L(M_q^{q_0}) \cdot (L(\underline{P}_{(q,p)}))^*$. Thus, we have proved that equation (i) holds. Similarly, we can prove $(L(\underline{P}_{(q,p)}))^+ \subseteq L(\underline{P}_{(q,p)})$ by the fact that $(L((A^q)_p^p))^+ \subseteq L((A^q)_p^p)$ and $L(\underline{P}_{(q,p)}) \subseteq L((A^q)_p^p)$. Together with the fact that $L(\underline{P}_{(q,p)}) \subseteq (L(\underline{P}_{(q,p)}))^+$, we conclude that equation (ii) holds.

Let $U, V \subseteq \Sigma^*$ be two languages such that $UV^* = U$ and $V^+ = V$. Then if $w' \in \mathrm{UP}(UV^\omega)$, there must exist two words $u \in U$ and $v \in V$ such that $w' = uv^\omega$ according to Lemma 3. We let $U = L(M_q^{q_0})$ and $V = L(\underline{P}_{(q,p)})$. According to equations (i) and (ii), we have that $UV^* = U$ and $V^+ = V$. Since $w \in L(UV^\omega)$, there must exist two words $x \in L(M_q^{q_0})$ and $y \in L(\underline{P}_{(q,p)})$ such that $w = x \cdot y^\omega$. In other words, w is accepted by $\mathcal{F}$. Thus $\mathrm{UP}(L(\underline{B})) \subseteq \mathrm{UP}(\mathcal{F})$ holds.

- $\mathrm{UP}(\mathcal{F}) \subseteq \mathrm{UP}(L(\overline{B}))$. Let w be an ω-word in $\mathrm{UP}(\mathcal{F})$. Then there exists a decomposition (u, v) of w such that $uv \backsim_M u$ and p is an accepting state of A^q where $q = M(u)$ and $p = A^q(v)$. It follows that $u \in L(M_q^{q_0})$ since $q = M(u)$. Further, we have $v \in L(\overline{P}_{(q,p)})$ since $\overline{P}_{(q,p)} = M_q^q \times (A^q)_p^{s_q}$ according to Definition 8 where s_q is the initial state of A^q. It follows that $u \cdot v^\omega \in L(M_q^{q_0}) \cdot (L(\overline{P}_{(q,p)}))^\omega \subseteq \mathrm{UP}(L(\overline{B}))$.
- $\mathrm{UP}(L(\underline{B})) = \mathrm{UP}(\mathcal{F})$ if $\mathcal{F}$ is a canonical FDFA. $\mathrm{UP}(L(\underline{B})) \subseteq \mathrm{UP}(\mathcal{F})$ holds for any FDFA $\mathcal{F}$. Thus, we only have to prove that $\mathrm{UP}(\mathcal{F}) \subseteq \mathrm{UP}(L(\underline{B}))$ if $\mathcal{F}$ is a canonical FDFA. It directly follows from Proposition 1 and Lemma 5.

Therefore, we complete the proof. ■

Lemma 8 introduces a special class of canonical FDFAs $\mathcal{F}$ for which the over-approximation method produces a BA $\overline{B}$ such that $\mathrm{UP}(\overline{B}) = \mathrm{UP}(\mathcal{F})$. In order to prove Lemma 8, we will first introduce Lemma 7, which has also been used to analyze the counterexamples from the BA teacher in Sect. 8.1.

Lemma 7. *Let $\mathcal{F} = (M, \{A^q\})$ be an FDFA and $\overline{B}$ be the BA constructed from $\mathcal{F}$ by Definition 8. For any $w \in UP(L(\overline{B}))$, there are a decomposition (u, v) of w and an integer $n \geq 1$ such that $v = v_1 \cdots v_n$ and for all $i \in [1 \cdots n]$, $v_i \in L(A^{M(u)})$ and $uv_i \backsim_M u$.*

Proof. Since we only consider ultimately periodic words of $L(\overline{B})$, the ω-words will be given by their decompositions in this proof. In the following we fix an ω-word w.

According to Definition 8, $\mathrm{UP}(L(\overline{B})) = \bigcup_{q \in Q, p \in F_q} L(M_q^{q_0}) \cdot (L(\overline{P}_{(q,p)}))^\omega$ where Q is the state set of M and F_q is the set of accepting states of A^q. It follows that w can be given by a decomposition (u, v) such that $u \in L(M_q^{q_0})$ and $v \in (L(\overline{P}_{(q,p)}))^+$ for some $p \in F_q$ where $q = M(u)$. Thus, there exists an integer $n \geq 1$ such that $v = v_1 \cdots v_n$ and $v_i \in L(\overline{P}_{(q,p)})$ for every $1 \leq i \leq n$. In addition, we have that $uv_i \backsim_M u$ and $v_i \in L((A^q)_p^{s_q})$ for every $1 \leq i \leq n$ since $\overline{P}_{(q,p)} = M_q^q \times (A^q)_p^{s_q}$ where s_q is the initial state in A^q. Note that p is the only accepting state of $(A^q)_p^{s_q}$ since $(A^q)_p^{s_q}$ is obtained from A^q by setting $p \in F_q$ as its only accepting state. Moreover, $p = (A^q)_p^{s_q}(v_i) = A^q(v_i)$ for every $1 \leq i \leq n$.

The remaining proof will show how we can find the accepting state p in A^q for w. Let w be a word given by the decomposition (u, v). From the decomposition, we can construct an FA $\mathcal{D}_{u\$v}$ such that $L(\mathcal{D}_{u\$v}) = \{u'\$v' \mid u'v'^{\omega} = uv^{\omega}\}$ by the method introduced later in Sect. 8.2 where \$ is not a letter in Σ. We note that the number of states in $\mathcal{D}_{u\$v}$ is in $O(|v|(|v| + |u|))$ (see Sect. 8.2). In addition, we can construct an FA $\mathcal{A}$ such that $L(\mathcal{A}) = \bigcup_{q\in Q, p\in F_u} L(M_q^{q_0}) \cdot \$ \cdot (L(M_q^q \times (A^q)_p^{s_q}))^+$. By fixing a state q of M and an accepting state p of A^q, we can construct an FA $\mathcal{A}_{(q,p)}$ such that $L(\mathcal{A}_{(q,p)}) = L(M_q^{q_0}) \cdot \$ \cdot (L(M_q^q \times (A^q)_p^{s_q}))^+ = L(M_q^{q_0}) \cdot \$ \cdot (L(\overline{P}_{(q,p)}))^+$. Recall that in the over-approximation construction, $\overline{P}_{(q,p)}$ is defined as $M_q^q \times (A^q)_p^{s_q}$. We can identify the corresponding q and p such that $L(\mathcal{A}_{(q,p)} \times \mathcal{D}_{u\$v}) \neq \emptyset$ since there must exist such two states q and p otherwise uv^{ω} will not be accepted by $\overline{B}$. To get all the fragment words v_i out of v, one only needs to use $\overline{P}_{(q,p)}$ to run the finite word v. The time and space complexity of this procedure are in $O(nk(n + nk) \cdot (|v|(|v| + |u|)))$ and $O((n + nk) \cdot (|v|(|v| + |u|)))$ respectively where n is the number of states of M and k is the number of states in the largest progress DFA of $\mathcal{F}$. Thus we complete the proof. ■

Lemma 8. *Let $\mathcal{F} = (M, \{A^q\})$ be a canonical FDFA and for every progress DFA A^q of $\mathcal{F}$, we have $v_1 \cdots v_n \in L(A^q)$ for any $v_i \subseteq \Sigma^*$ and $n \geq 1$ if $p = A^q(v_1) = \cdots = A^q(v_n)$ and p is an accepting state of A^q. Then $UP(\mathcal{F}) = UP(L(\overline{B}))$ where $\overline{B}$ is the BA constructed from $\mathcal{F}$ by the over-approximation method in Definition 8.*

Proof. According to Lemma 6, we have $\text{UP}(\mathcal{F}) \subseteq \text{UP}(L(\overline{B}))$. According to Lemma 7, an ω-word $w \in \text{UP}(L(\overline{B}))$ can be given by a decomposition $(u, v_1 \cdots v_n)$ for some $n \geq 1$ such that for $i \in [1 \cdots n]$, $v_i \in L(A^{M(u)})$ and $uv_i \backsim_M u$. By the assumption, we have that $v_1 \cdots v_n \in L(A^{M(u)})$. It follows that $uv_1 \cdots v_n \backsim_M u$ and $v_1 \cdots v_n \in L(A^{M(u)})$, which indicates that $u(v_1 \cdots v_n)^{\omega}$ is accepted by $\mathcal{F}$. Therefore, $w \in \text{UP}(\mathcal{F})$ and it follows that $\text{UP}(L(\overline{B})) \subseteq \text{UP}(\mathcal{F})$. Thus, we complete the proof. ■

It is easy to see that all three canonical FDFAs depicted in Fig. 2 satisfy Lemma 8. Therefore, we can also use over-approximation method to construct the BAs accepting $a^{\omega} + ab^{\omega}$ from them.

7.1. From FDFAs to Limit Deterministic Büchi Automata

Recall that in the under-approximation (respectively, over-approximation) method, we need first construct an FA $\underline{P}_{(u,v)} = M_u^u \times (A^u)_v^{s_u} \times (A^u)_v^v$ (respectively $\overline{P}_{(u,v)} = M_u^u \times (A^u)_v^{s_u}$) and then construct an NBA recognizing $L(\underline{P}_{(u,v)})^{\omega}$ (respectively, $L(\overline{P}_{(u,v)})^{\omega}$).

In this section, we show that we can construct a DBA A instead of a BA defined in Definition 8 recognizing $L(\underline{P}_{(u,v)})^{\omega}$ (respectively, $L(\overline{P}_{(u,v)})^{\omega}$), which yields a limit deterministic Büchi automaton from the conjectured FDFA $\mathcal{F}$.

To make our construction more general, in the following we construct a DBA A with $L(A) = L(D)^{\omega}$ from a DFA D with only one accepting state. One can check that the FAs $\underline{P}_{(u,v)}$ and $\overline{P}_{(u,v)}$ from the under-approximation and the over-approximation methods indeed are DFAs with one accepting state.

Definition 9. Let $D = (\Sigma, Q, q_0, \delta, \{q_f\})$ be a DFA in which every state $q \in Q$ can be reached by q_0 and can reach q_f. The DBA A is defined as the tuple $(\Sigma, Q', q_0, \delta', \{[q_f]\} \cup \{(q_f, q) \mid q \in Q\})$ where $Q' = Q \cup (Q \times Q) \cup \{[q] \mid q \in Q\} \cup \{\langle q\rangle \mid q \in Q\}$ and δ' is defined as follows where $q, q' \in Q$ and $a \in \Sigma$:

1.
$$\delta'(q,a) = \begin{cases} \delta(q,a) & q \neq q_f; \\ (\delta(q_0,a), \delta(q_f,a)) & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) \neq \emptyset; \\ [\delta(q_0,a)] & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) = \emptyset; \\ \langle \delta(q_f,a) \rangle & q = q_f \wedge \delta(q_0,a) = \emptyset \wedge \delta(q_f,a) \neq \emptyset. \end{cases}$$

2.
$$\delta'((q,q'),a) = \begin{cases} (\delta(q,a), \delta(q',a)) & q \neq q_f \wedge \delta(q,a) \neq \emptyset \wedge \delta(q',a) \neq \emptyset; \\ \langle \delta(q',a) \rangle & q \neq q_f \wedge \delta(q,a) = \emptyset \wedge \delta(q',a) \neq \emptyset; \\ [\delta(q,a)] & q \neq q_f \wedge \delta(q,a) \neq \emptyset \wedge \delta(q',a) = \emptyset; \\ (\delta(q_0,a), \delta(q',a)) & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q',a) \neq \emptyset; \\ \langle \delta(q',a) \rangle & q = q_f \wedge \delta(q_0,a) = \emptyset \wedge \delta(q',a) \neq \emptyset; \\ [\delta(q_0,a)] & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q',a) = \emptyset. \end{cases}$$

3.
$$\delta'(\|q\|,a) = \begin{cases} \|\delta(q,a)\| & q \neq q_f; \\ (\delta(q_0,a), \delta(q_f,a)) & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) \neq \emptyset; \\ [\delta(q_0,a)] & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) = \emptyset; \\ \langle \delta(q_f,a) \rangle & q = q_f \wedge \delta(q_0,a) = \emptyset \wedge \delta(q_f,a) \neq \emptyset; \end{cases}$$

where $\|\cdot\|$ is either $\langle\cdot\rangle$ or $[\cdot]$ in the above definition.

Note that the transition function δ may not be complete, i.e., it happens that $\delta(q,a) = \emptyset$ for some $q \in Q$ and $a \in \Sigma$. The reason why we only keep the states of D which can be reached by q_0 and can reach q_f rather than considering a complete DFA is that: (1) the number of states in A can be reduced in this way since $|Q'|$ is quadratic in the number of states of D; (2) it is simple for the construction to identify the words that cannot be extended to a word in $L(D)$ by observing that their corresponding runs cannot extend any more. This is because we want A to accept $L(D)^\omega$ and the words which cannot be extended to words in $L(D)$ are not valid prefixes of words in $L(D)^\omega$ and thus can be omitted in the construction. We omit in the definition of δ' the cases where there are no successor states on those transitions. One can check that the definition δ' is well defined in the sense that all possible situations are taken account of.

Let $U = L(D)$ and $K = L(D^{q_f}_{q_f})$ where $D^{q_f}_{q_f}$ is obtained from D by setting q_f as the initial state and the accepting state. We divide the language $U \cup K$ into three parts, namely $U \setminus K = \{u \in \Sigma^* \mid u \in U \wedge u \notin K\}$, $U \cap K = \{u \in \Sigma^* \mid u \in U \wedge u \in K\}$ and $K \setminus U = \{u \in \Sigma^* \mid u \notin U \wedge u \in K\}$. We will explain the roles of the three languages in the construction below.

Let us consider the run r of A over ω-word u^ω where $u \in U$. After reading u, r reaches the accepting state q_f and still wants to continue the run. Our goal is to extend the run r so that it can visit an accepting state of A infinitely often. Since $u \in U$, we can then extend the run r by going from the state q_0 to the state q_f again. We try to make r starting from q_0 again by starting at (q_0, q_f). The first element q_0 of the state pair performs the behaviors starting at the initial state q_0, while the second element q_f tries to mimic the behaviors starting from the accepting state q_f. If $u \in U \cap K$, then r can reach an accepting state (q_f, q_f) by starting at state (q_0, q_f). A finite word $u \in U \setminus K$ may be detected on the run r by observing that some state $(q_1, q_2) = \delta'((q_0, q_f), u_1)$ on the run such that $\delta(q_1, a) \neq \emptyset$ while $\delta(q_2, a) = \emptyset$ where $u = u_1 \cdot a \cdot u_2$. Thus, in order to track the remaining behavior of r on the word u_2, i.e., the behavior of the left part of the state pair, we make use of states of the form of $[q]$. Let $[q_1'] = \delta'((q_1, q_2), a)$. We have $[q_f] = \delta'([q_1'], u_2)$ since

$u \in U \setminus K$. In other words, we use the states in form of $[q]$ to track the behavior of the left part of the state pair if the behavior of the right part disappears. Similarly, a finite word $u = u_1 \cdot a \cdot u_2 \in U$ with $u_1 \in U \cap K$ and $a \cdot u_2 \in K \setminus U$ may be detected in a symmetric way. $(q_f, q_f) = \delta'((q_0, q_f), u_1)$ since $u_1 \in U \cap K$. We can assume that $\delta(q_0, a) = \emptyset$ while $\delta(q_f, a) \neq \emptyset$ since $a \cdot u_2 \in K \setminus U$. Let $\langle q'_2 \rangle = \delta'((q_f, q_f), a)$. In such a case, we use the states in form of $\langle q \rangle$ to keep track of the rest behavior of r on u_2, i.e., the behavior of the right part of the state pair. We remark that one reason to distinguish between the states in form of $[q]$ and of $\langle q \rangle$ is the acceptance condition since $[q_f]$ is an accepting state while $\langle q_f \rangle$ is not. The other reason to distinguish them is that when A is reading an accepting ω-word w, we can tell that A is reading a fragment word $u \in U \setminus K$ of w if a state $[q]$ occurs on the run of A while A is reading a fragment word $u \in K \setminus U$ if a state $\langle q \rangle$ occurs on the run.

In the construction, every time when a run encounters the states $[q_f]$, (q_f, q) and $\langle q_f \rangle$, we try to make it mimic the behavior of q_0 if possible. In this way, any ω-word w accepted by A can be rewritten as the concatenation of infinitely many finite words accepted by D, i.e., $w = u_1 \cdot u_2 \cdots$ with $u_i \in L(D)$ for any $i \geq 1$.

Theorem 2. *Let D be a DFA with one accepting state and A be the DBA constructed from D by Definition 9. Then $L(A) = L(D)^\omega$.*

Proof. Recall that $U = L(D)$ and $K = L(D^{q_f}_{q_f})$. According to Theorem 1, we only consider ultimately periodic words of ω-regular languages.

1. $\mathrm{UP}(U^\omega) \subseteq \mathrm{UP}(L(A))$. Let $w \in \mathrm{UP}(U^\omega)$. w can be written as $v \cdot (u_0 \cdot u_1 \cdots u_i \cdots u_n)^\omega$ for some $n \geq 0$ where $v \in U^*$ and $u_i \in U$ for any $0 \leq i \leq n$. The goal is to prove that the corresponding run r of A over w visits $[q_f]$ or (q_f, q) for infinitely many times for some state $q \in Q$. According to the definition of δ', any state $q \in Q$ will not be reached again once r touches the accepting state q_f. Starting from q_0 after reading $v \cdot u_0 u_1$, it will either stop at the state $[q_f]$, $\langle q_f \rangle$ or state (q_f, q) for some $q \in Q$. In the following, we show that starting from state $[q_f]$, $\langle q_f \rangle$ or state (q_f, q), it will visit (q_f, q) or $[q_f]$ at least once and then stop at either $[q_f]$, $\langle q_f \rangle$ or state (q_f, q) after reading arbitrary $u \in U$.

   - If $u \in U \cap K$, then $\delta'([q_f], u) = \delta'(\langle q_f \rangle, u) = \delta'((q_0, q_f), u) = (q_f, q_f)$ and $\delta'((q_f, q), u) = (q_f, p)$ for some $p, q \in Q$ according to Definition 9.
   - Otherwise $u \in U \setminus K$. According to Definition 9, upon reading the word u, the run r at the state $[q_f]$ or the state $\langle q_f \rangle$ will restart at (q_0, q_f), while the run r at (q_f, q) will restart at (q_0, q). After reading u, the run r will either stop at $[q_f]$ or (q_f, q') for some $q' \in Q$. The reason why this happens is that starting from the first element q_0 of (q_0, q_f) and (q_0, q), the run r will not visit any states in form of $\langle q \rangle$ since $u \in U \setminus K$.

   Therefore, with infinitely many $u \in U$ in w, the run r will visit some accepting state in A infinitely often, which implies that $w \in \mathrm{UP}(L(A))$.
2. $\mathrm{UP}(L(A)) \subseteq \mathrm{UP}(U^\omega)$. Let w be an ultimately periodic word accepted by A. We assume that one of its corresponding run r is in the form of $q_0 \xrightarrow{u_0} q_1 \xrightarrow{u_1} q_2 \xrightarrow{u_2} \cdots q_h \xrightarrow{u_h} q_{h+1} \xrightarrow{u_{h+1}} \cdots q_{n-1} \xrightarrow{u_{n-1}} q_n \xrightarrow{u_n} q_h$ where $u_i \in \Sigma^+$ and $q_i \in \{(q_f, p) \mid p \in Q\} \cup \{[q_f], \langle q_f \rangle\}$ for every $1 \leq i \leq n$. We also assume that q_h is the first accepting state that occurs on the run r infinitely often where $1 \leq h \leq n$. We note that there does not exist a state q that belongs to the set $\{(q_f, p) \mid p \in Q\} \cup \{[q_f], \langle q_f \rangle\}$ on the fragment run from q_i to q_{i+1} for any $i \geq 0$. Since w is accepted by A, we have that $q_h \in \{[q_f], (q_f, q)\}$ for some state $q \in Q$.

The goal is to prove that w can be written as the form of $v_0v_1v_2\cdots v_d(v_{d+1}\cdots v_m)^\omega$ for some $m \geq d \geq 0$ such that $v_i \in U$ for every $0 \leq i \leq m$. In the following, we explain why it is possible to find those finite words $v \in U$ from the given run r.

- There is a fragment run of r from a state q_i to a state $q_{i+1} \in \{[q_f]\} \cup \{([q_f], q) \mid q \in Q\}$ over the finite word u_i, which is depicted as follows.

$$q_i \xrightarrow{u_i} [q_f] \text{ or } q_i \xrightarrow{u_i} ([q_f], q) \tag{1}$$

  In this situation, we let $v = u_i$ and according to the construction, u_i is accepted by D, i.e., $v \in U$.

- If there is a fragment run of r from a state q_i to the state $q_{i+1} = \langle q_f\rangle$ on the finite word u_i, we will find all consecutive states $\langle q_f\rangle$ behind q_{i+1}, which is depicted as follows.

$$q_i \xrightarrow{u_i} q_{i+1} = \langle q_f\rangle \xrightarrow{u_{i+1}} \cdots \xrightarrow{u_{j-1}} q_j = \langle q_f\rangle \xrightarrow{u_j} q_{j+1} \in \{[q_f], (q_f, q)\} \tag{2}$$

  In this case, we let $v = u_i \cdot u_{i+1} \cdots u_{j-1}$. It is easy to verify that $v \in U$ according to the construction.

Therefore, it follows that we can rewrite the ω-word w into a word which is clearly in U^ω. Therefore, for any ultimately periodic word $u_0(u_1)^\omega \in \mathrm{UP}(L(A))$, there exists an integer $n \geq 0$ such that $u_0(u_1)^\omega = v_0 \cdots v_i(v_{i+1}\cdots v_n)^\omega$ and $v_i \in U$ for every $0 \leq i \leq n$.

Therefore we complete the proof. ■

Figure 12 gives an example for the DBA A constructed from the DFA D.

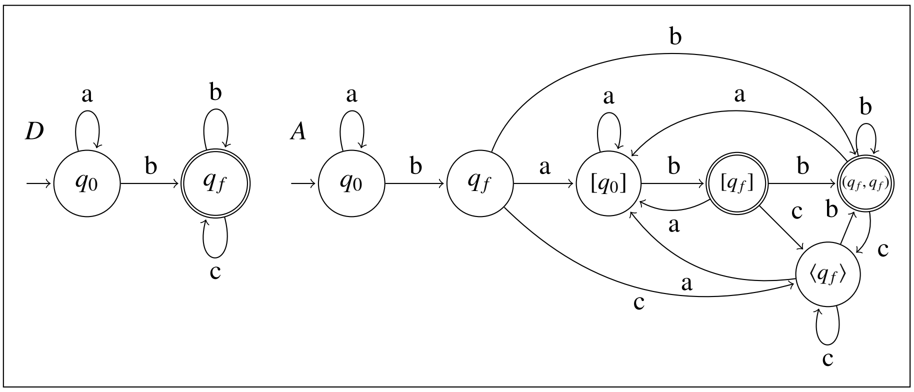


Figure 12: An example for the LDBA construction

Lemma 9 (Size of the DBA). *Let D be a DFA with one accepting state and a set Q of states. If A is the DBA constructed from D by Definition 9, then the number of states in A is in $O(|Q|^2)$.*

Proof. The proof is trivial since the state set Q' of A is defined as $Q \cup (Q \times Q) \cup \{[q] \mid q \in Q\} \cup \{\langle q\rangle \mid q \in Q\}$. ■

Corollary 1 (Sizes of the LDBAs). *Let $\mathcal{F} = (M, \{A^u\})$ be an FDFA and $\underline{B}$, $\overline{B}$ be the LDBAs constructed from $\mathcal{F}$ by replacing the BAs in Definition 8 with the DBAs in Definition 9. Let n and k be the numbers of states in M and the largest progress DFA of $\mathcal{F}$, respectively. Then the numbers of states of $\underline{B}$ and $\overline{B}$ are in $O(n^3k^5)$ and $O(n^3k^3)$, respectively.*

Proof. According to Definition 8, the sizes of $\underline{P}_{(u,v)}$ and $\overline{P}_{(u,v)}$ are in $O(nk^2)$ and $O(nk)$ respectively. Thus the sizes of DBAs recognizing $\underline{N}_{(u,v)}$ and $\overline{N}_{(u,v)}$ are in $O(n^2k^4)$ and $O(n^2k^2)$ respectively according to Lemma 9. Moreover, the number of the pairs (u, v) are at most nk. Thus we complete the proof. ■

Corollary 2 (Languages of the LDBAs). *Let $\mathcal{F}$ be an FDFA and $\underline{B}_d$, $\overline{B}_d$ be the LDBAs constructed from $\mathcal{F}$ by replacing the BAs in Definition 8 with DBAs in Definition 9. Let $\underline{B}$ and $\overline{B}$ be the BAs constructed from $\mathcal{F}$ by Definition 8. Then we have that $L(\underline{B}) = L(\underline{B}_d)$ and $L(\overline{B}) = L(\overline{B}_d)$.*

Proof. The proof directly follows the construction for LDBAs. ■

8. Counterexample Analysis for the FDFA Teacher

In this section, we first show how to extract valid counterexamples for the FDFA learner from the counterexamples returned from the BA teacher and then give their correctness proofs in Sect. 8.1. Since the counterexample analysis procedure makes use of three DFAs, namely $\mathcal{D}_{u\$v}$, $\mathcal{D}_1$ and $\mathcal{D}_2$ (see Sect. 8.1), we will give the DFA construction for $\mathcal{D}_{u\$v}$ in Sect. 8.2 and the constructions for $\mathcal{D}_1$ and $\mathcal{D}_2$ in Sect. 8.3.

8.1. Counterexample Analysis

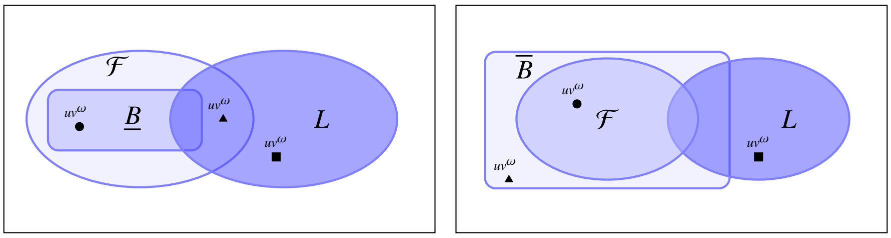
During the learning procedure, if we failed the equivalence query for the BA B, the BA teacher will return a counterexample uv^ω to the FDFA teacher.

Definition 10 (Counterexample for the FDFA Teacher). Let $B \in \{\underline{B}, \overline{B}\}$ be the conjectured BA and L be the target language. We say that the counterexample uv^ω is

- *positive* if $uv^\omega \in \mathrm{UP}(L)$ and $uv^\omega \notin \mathrm{UP}(L(B))$, and
- *negative* if $uv^\omega \notin \mathrm{UP}(L)$ and $uv^\omega \in \mathrm{UP}(L(B))$.

Obviously, the above definition is different to the counterexample for the FDFA learner in Definition 7. Below we illustrate the necessity of the counterexample analysis procedure by an example.

Example 3. *Again, consider the conjectured FDFA $\mathcal{F}$ depicted in Fig. 1 for $L = a^\omega + b^\omega$. Suppose the BA teacher returns a negative counterexample $(ba)^\omega$. In order to remove $(ba)^\omega \in UP(\mathcal{F})$, one has to find a decomposition of $(ba)^\omega$ that $\mathcal{F}$ accepts, which is the goal of the counterexample analysis. Not all decompositions of $(ba)^\omega$ are accepted by $\mathcal{F}$. For instance, (ba, ba) is accepted while (bab, ab) is not.*



(a) Under-Approximation

(b) Over-Approximation

Figure 13: The Case for Counterexample Analysis

A positive (respectively negative) counterexample uv^ω for the FDFA teacher is *spurious* if $uv^\omega \in \mathrm{UP}(\mathcal{F})$ (respectively $uv^\omega \notin \mathrm{UP}(\mathcal{F})$). Suppose we use the under-approximation method to construct the BA $\underline{B}$ from $\mathcal{F}$ depicted in Fig. 10. The BA teacher returns a spurious positive counterexample b^ω, which is in $\mathrm{UP}(\mathcal{F})$ but not in $\mathrm{UP}(L(\underline{B}))$. We show later that in such a case, one can always find a decomposition, in this example (b, bb), as the counterexample for the FDFA learner.

Let $\mathcal{F} = (M, \{A^u\})$ be the current conjectured FDFA. In order to analyze the returned counterexample uv^ω, we define three DFAs below:

- a DFA $\mathcal{D}_{u\$v}$ with $L(\mathcal{D}_{u\$v}) = \{u'\$v' \mid u' \in \Sigma^*, v' \in \Sigma^+, uv^\omega = u'v'^\omega\}$,
- a DFA $\mathcal{D}_1$ with $L(\mathcal{D}_1) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \backsim_M u, v \in L(A^{M(u)})\}$, and
- a DFA $\mathcal{D}_2$ with $L(\mathcal{D}_2) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \backsim_M u, v \notin L(A^{M(u)})\}$.

Here \$ is a letter not in Σ. The constructions for the three DFAs will be introduced in Sect. 8.2 and Sect. 8.3. Intuitively, $\mathcal{D}_{u\$v}$ accepts every possible decomposition (u', v') of uv^ω; $\mathcal{D}_1$ accepts every decomposition (u', v') that is accepted by $\mathcal{F}$; and $\mathcal{D}_2$ accepts every decomposition (u', v') that is not accepted by $\mathcal{F}$ yet $u'v' \backsim_M u'$.

We will use different counterexample analysis procedures for the under-approximation method and the over-approximation method of the translation from FDFAs to BAs since the analysis for the returned counterexample has to first identify which kind of the counterexample is as in Fig. 13 and then deals with it accordingly as follows.

Let $\underline{B}$ be the BA constructed from $\mathcal{F}$ by the under-approximation method. Recall that $\mathrm{UP}(L(\underline{B})) \subseteq \mathrm{UP}(\mathcal{F})$. Figure 13(a) depicts all possible cases of $uv^\omega \in \mathrm{UP}(L(\underline{B})) \ominus \mathrm{UP}(L)$ by means of different shapes.

U1 : $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \notin \mathrm{UP}(\mathcal{F})$ (square). The word uv^ω is a positive counterexample; one has to find a decomposition (u', v') of uv^ω such that $u'v' \backsim_M u'$ and $v' \in L(A^{M(u')})$. This can be easily done by taking a word $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$.

U2 : $uv^\omega \notin \mathrm{UP}(L) \wedge uv^\omega \in \mathrm{UP}(\mathcal{F})$ (circle). The word uv^ω is a negative counterexample; one needs to find a decomposition (u', v') of uv^ω that is accepted by $\mathcal{F}$. This can be done by taking a word $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$.

U3 : $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \in \mathrm{UP}(\mathcal{F})$ (triangle). The word uv^ω is a spurious positive counterexample. Since $uv^\omega \in \mathrm{UP}(\mathcal{F})$, there must exist one decomposition of uv^ω accepted by $\mathcal{F}$, say

(u, v). According to Lemma 5, there must exist some $k \geq 1$ such that (u, v^k) is not accepted by $\mathcal{F}$. Therefore, we can also use the same method in U1 to get a counterexample (u', v').

We can also use the over-approximation method to construct a BA $\overline{B}$ from $\mathcal{F}$ with $\mathrm{UP}(\mathcal{F}) \subseteq \mathrm{UP}(L(\overline{B}))$. All possible cases for the counterexample $uv^\omega \in \mathrm{UP}(L(\overline{B})) \ominus \mathrm{UP}(L)$ is depicted in Fig. 13(b) by means of different shapes.

O1 : $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \notin \mathrm{UP}(\mathcal{F})$ (square). The word uv^ω is a positive counterexample that can be dealt with the same method for case U1.

O2 : $uv^\omega \notin \mathrm{UP}(L) \wedge uv^\omega \in \mathrm{UP}(\mathcal{F})$ (circle). The word uv^ω is a negative counterexample that can be dealt with the same method for case U2.

O3 : $uv^\omega \notin \mathrm{UP}(L) \wedge uv^\omega \notin \mathrm{UP}(\mathcal{F})$ (triangle). Here uv^ω is a spurious negative counterexample. In such a case, it is possible that we cannot find a valid decomposition out of uv^ω to refine $\mathcal{F}$. According to Lemma 7, we may find a decomposition (u', v') of uv^ω and an integer $n \geq 1$ such that $v' = v_1 v_2 \cdots v_n$ and for each $1 \leq i \leq n$, $u'v_i \backsim_M u'$ and $v_i \in L(A^{M(u')})$. It follows that (u', v_i) is accepted by $\mathcal{F}$ for every $1 \leq i \leq n$. If we find some $i \in [1 \cdots n]$ such that $u'v_i^\omega \notin \mathrm{UP}(L)$, then we return (u', v_i), otherwise, we terminate the learning procedure and report we are not able to find a suitable counterexample to refine the current FDFA.

From an implementation point of view, we note that determining whether $uv^\omega \in \mathrm{UP}(L)$ can be done by posing a membership query $\mathsf{Mem}^{\mathsf{BA}}(uv^\omega)$, and checking whether $uv^\omega \in \mathrm{UP}(\mathcal{F})$ boils down to checking the emptiness of $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$.

Lemma 10. *Let uv^ω be the counterexample returned by the BA teacher. For the under-approximation method, we can always return a valid counterexample (u', v') for the FDFA learner. For the over-approximation method, if counterexample analysis returns a decomposition (u', v'), then it is a valid counterexample for the FDFA learner.*

Proof. Let M be the leading DFA of the current conjectured FDFA $\mathcal{F}$.

- Case U1 and O1: $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \notin \mathrm{UP}(\mathcal{F})$. According to Definition 7, uv^ω is a positive counterexample. We need to return a positive counterexample (u', v') for the FDFA learner in order to make the conjectured FDFA to accept it. The counterexample (u', v') can be found by taking a word $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$. We first need to prove that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$ is not empty. Since $uv^\omega \notin \mathrm{UP}(\mathcal{F})$, we know that all decompositions of uv^ω, such as (u, v), are not accepted by $\mathcal{F}$. According to [1], we can always find a normalized factorization (x, y) of (u, v) with respect to M where $x = uv^i$ and $y = v^j$ from some $0 \leq i < j$ such that $xy \backsim_M x$. Therefore (x, y) is also a decomposition of uv^ω and it is not accepted by $\mathcal{F}$. In other words, $y \notin L(A^q)$ where $q = M(x)$ and $xy \backsim_M x$. It follows that $x\$y \in L(\mathcal{D}_2)$ according to Proposition 5 (introduced in Sect. 8.3). Thus, we conclude that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$ is not empty. We let $u' = x$ and $v' = y$, and it is easy to verify that (u', v') is a positive counterexample for FDFA learner.
- Case U3: $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \in \mathrm{UP}(\mathcal{F})$. In this case, uv^ω is a spurious positive counterexample, which happens when we use the under-approximation method to construct BAs. We need to return a positive counterexample (u', v') for the FDFA learner in order to make the conjectured FDFA to accept it. The counterexample (u', v') can be also found by taking

a word $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$. Again, we need to prove that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$ is not empty. Since $uv^\omega \in \text{UP}(\mathcal{F})$, there exists some decomposition of uv^ω, say (u, v), which is accepted by $\mathcal{F}$. According to Lemma 5, there exists some $k \geq 1$ such that (u, v^k) is not accepted by $\mathcal{F}$ since $uv^\omega \notin \text{UP}(L(\underline{B}))$. Moreover, $uv \backsim_M u$ since (u, v) is accepted by $\mathcal{F}$. It follows that $uv^k \backsim_M u$, which indicates that $u\$v^k \in L(\mathcal{D}_2)$ according to Proposition 5 (introduced in Sect. 8.3). Therefore, we conclude that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$ is not empty and every decomposition (u', v') taken from $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$ is a positive counterexample for FDFA learner.

- Case U2 and O2: $uv^\omega \notin \text{UP}(L) \wedge uv^\omega \in \text{UP}(\mathcal{F})$. In this case, uv^ω is a negative counterexample. One has to return a counterexample (u', v') such that $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$ for the FDFA learner to make the conjectured FDFA to reject it. We first need to prove that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$ is not empty. Since $uv^\omega \in \text{UP}(\mathcal{F})$, there exists some decomposition (u', v') of uv^ω that is accepted by $\mathcal{F}$. It follows that $u'\$v' \in L(\mathcal{D}_1)$ according to Proposition 4. Thus we conclude that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$ is not empty. Moreover, it is easy to verify that (u', v') is a negative counterexample for the FDFA learner.

- Case O3: $uv^\omega \notin \text{UP}(L) \wedge uv^\omega \notin \text{UP}(\mathcal{F})$. In this case, uv^ω is a spurious negative counterexample, which happens when we use the over-approximation method to construct BAs. It is possible that we cannot find a valid decomposition (u', v') to refine $\mathcal{F}$. According to the proof of Lemma 7, one can construct a decomposition (u, v) of uv^ω such that $v = v_1 \cdot v_2 \cdots v_n$ for some $n \geq 1$ and for each $i \in [1 \cdots n]$, we have that $v_i \in L(A^{M(u)})$ and $uv_i \backsim_M u$. Therefore, we let $u' = u$ and $v' = v_i$ if there exists some $i \geq 1$ such that $uv_i^\omega \notin \text{UP}(L)$. Clearly, (u', v') is a negative counterexample for the FDFA learner. ■

8.2. From an ω-word uv^ω to the DFA $\mathcal{D}_{u\$v}$

In [37], Calbrix *et al.* presented a canonical representation $L_\$ = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega \in L\}$ for a given ω-regular language L. In principle, we can apply their method to obtain the DFA $\mathcal{D}_{u\$v}$ from the given ω-word uv^ω. However, the number of states in their constructed DFA is in $O(2^{|u|+|v|})$. In this section, we present a more effective method to build the DFA $\mathcal{D}_{u\$v}$ such that $L(\mathcal{D}_{u\$v}) = \{u'\$v' \mid u' \in \Sigma^*, v' \in \Sigma^+, u'v'^\omega = uv^\omega\}$ for a given ω-word uv^ω. Moreover, the number of states in our constructed DFA $\mathcal{D}_{u\$v}$ is in $O(|v|(|v| + |u|))$. A similar construction for the DFA $\mathcal{D}_{u\$v}$ has been proposed in [36], which first computes a regular expression to represent all possible decompositions of uv^ω and then constructs the DFA $\mathcal{D}_{u\$v}$ from the regular expression. Compared to the construction in [36], ours is a direct construction from uv^ω to $\mathcal{D}_{u\$v}$ and we also give the complexity of the construction.

We first give an example DFA $\mathcal{D}_{u\$v}$ for the ω-word $(ab)^\omega$ in Fig. 14. We can see that both the decomposition (aba, ba) and the decomposition $(ababa, bababa)$ have the same suffix $(ba)^\omega$, which hints us that the second element of a decomposition can be simplified as long as we do not change the periodic word.

In the following, we denote by $u \trianglelefteq v$ to represent that u is a prefix of v, i.e., there exists some $1 \leq j \leq |v|$ such that $u = v[1 \cdots j]$. We denote by $u \triangleleft v$ if $u \trianglelefteq v$ and $u \neq v$. We give the notion of the *smallest period* of an ω-word $w = uv^\omega$ as follows.

Definition 11 (Smallest period). Let w be an ω-word given by (u, v). We say r is the smallest period of (u, v) if $r \trianglelefteq v$, $r^\omega = v^\omega$ and for any $t \triangleleft r$, $t^\omega \neq r^\omega$.

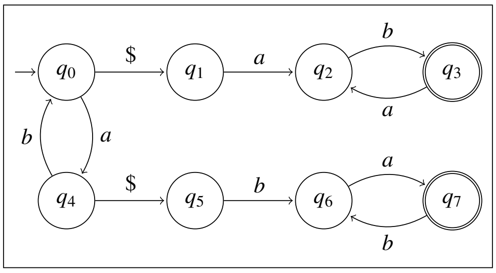


Figure 14: The DFA $\mathcal{D}_{u\$v}$ for ω-word (aba, ba)

Take $(ab)^\omega$ as an example: ab and ba are smallest periods of the decomposition (ab, ab) and the decomposition (aba, ba) respectively. We observe that $|ab| = |ba|$ and ab can be transformed to ba by shifting the first letter of ab to its tail. Formally, we prove in Lemma 11 that given an ω-word w, the length of its smallest periods is a constant number no matter what decomposition of w is given.

Lemma 11. *Let w be an ω-word given by two different decompositions (u, v) and (x, y). Let the smallest periods of (u, v) and (x, y) be r and t, respectively. Then either there exists an integer $j \geq 2$ such that $r = t[j \cdots n] \cdot t[1 \cdots j-1]$ with $|t| = n$ or $r = t$ holds.*

Proof. According to Definition 11, $w = uv^\omega = ur^\omega = xy^\omega = xt^\omega$. We prove it by contradiction. Without loss of generality, we assume that $|r| > |t|$. If $|u| = |x|$, we conclude that $r^\omega = t^\omega$ since $ur^\omega = xt^\omega$. It follows that r is not the smallest period of (u, v) since $t \lhd r$. Hence $|r| > |t|$ cannot hold in this case. Otherwise if $|u| \neq |x|$, we can either prove that $r = t$ or find some $j \geq 2$ such that $z = t[j \cdots n] \cdot t[1 \cdots j-1] \lhd r$ and $z^\omega = r^\omega$ in following cases, which also leads to contradiction. In the following two cases, $z = t$ if $k = 1$ and otherwise let $j = k$.

- $|u| > |x|$. Let $k = (|u| - |x|) \oplus |t| + 1$.
- $|x| > |u|$. Let $k = (|r| - (|x| - |u|) \oplus |r|) \oplus |t| + 1$.

Recall that $\oplus$ is the standard modular arithmetic operator. We depict the case when $|u| > |x|$ as follows, where $m = |u|$.

$$\begin{array}{ll} (u, r) & u[1]u[2] \cdots u[k]u[k+1] \cdots u[m] \cdot r \cdot r \cdot r \cdots \\ (x, t) & x[1]x[2] \cdots x[k]t[1] \cdots \cdot t[j-1] \cdot z \cdot z \cdot z \cdots \end{array}$$

We have that $z \lhd r$ by the assumption $|t| < |r|$. However, since $z^\omega = r^\omega$, we conclude that r is not the smallest period of (u, v), which leads to contradiction. Thus we complete the proof. ■

Lemma 11 shows that if the length of the smallest periods of an ω-word w is n, then there are exactly n different smallest periods for w. In the following, we define the *shortest form* for a decomposition of an ω-word.

Lemma 12. *Let w be an ω-word given by (u, v) and y be the smallest period of (u, v). Then there exist some $i \geq 0, j \geq 1$ and a word x such that $u = xy^i$ and $v = y^j$. Moreover, for any $x' \unlhd u$ such that $u = x'y^k$ for some $0 \leq k \leq i$, we have $x' = xy^{i-k}$. We say such (x, y) is the shortest form for (u, v).*

Proof. This lemma can be proved by Definition 11 along with the fact that $y^\omega = v^\omega$. In the following we will provide the procedure to construct (x, y). To find the shortest form of (u, v), we need to first find the smallest period y of (u, v), which is illustrated as follows. At first we initialize $k = 1$ and then do the following procedure: Let $y = v[1 \cdots k]$ and we check whether there exists some $1 \leq j \leq |v|$ such that $v = y^j$; we return y as the smallest period if we successfully find such an integer j; otherwise we increase k by 1 and repeat the above procedure. It is easy to verify that the returned y must be the smallest period of (u, v) such that $v^\omega = y^\omega$.

We will find the word x of the shortest form in the following procedure: Let $x = u$ and we will return ϵ if $x = \epsilon$ or $x = y$; otherwise we check whether there exists some $1 \leq k \leq |x|$ such that $x = x[1 \cdots k] \cdot y$; if there does not exist such k, we return x as the final result and otherwise we set u to $x[1 \cdots k]$ and repeat the above procedure. It follows that x is the shortest prefix of u such that $u = xy^i$ for some $i \geq 0$. ■

Corollary 3 is an immediate result of Lemma 12.

Corollary 3. *Let (u_1, v_1) and (u_2, v_2) be two decompositions of the ω-word uv^ω. If (u_1, v_1) and (u_2, v_2) share the smallest period y, then they also have the same shortest form (x, y) such that $u_1 = xy^i, u_2 = xy^j$ for some $i, j \geq 0$.*

Proof (Sketch). The assumption that (u_1, v_1) and (u_2, v_2) have different shortest forms will lead to the contradiction that $u_1(v_1)^\omega \neq u_2(v_2)^\omega$. ■

According to Corollary 3, all decompositions of an ω-word w that share the smallest period y can be represented as (xy^i, y^j) for some $i \geq 0, j \geq 1$. In addition, the number of different shortest forms of w is $|y|$ since there are $|y|$ different smallest periods according to Lemma 11. Thus we can denote the set of decompositions of w by the set $\bigcup_{k=1}^{|y|}\{(x_k y_k^i, y_k^j) \mid i \geq 0, j \geq 1\}$ where (x_k, y_k) is the k-th shortest form of w. It follows the construction of $\mathcal{D}_{u\$v}$ introduced below.

8.2.1. The Construction of the DFA $\mathcal{D}_{u\$v}$

Let w be an ω-word given by (u, v) and (x, y) be the shortest form of (u, v). In order to construct the DFA $\mathcal{D}_{u\$v}$, we assume that there is an algorithm constructing a DFA D such that $L(D) = xy^*\$y^+$ for the shortest form (x, y). Let $n = |y|$. We can construct the DFA $\mathcal{D}_{u\$v}$ such that $L(\mathcal{D}_{u\$v}) = \bigcup_{i=1}^n L(D_i)$ and $L(D_i) = x_i y_i^* \$ y_i^+$ for every $1 \leq i \leq n$ where $x_i = x \cdot y[1 \cdots i]$ and $y_i = y[i + 1 \cdots n] \cdot y[1 \cdots i]$.

Let $m = |x|$. We introduce how to construct the DFA D that accepts $xy^*\$y^+$ below.

- If $m = 0$, we define the DFA D as the tuple $(\Sigma, \{q_0, \ldots, q_{2n}\}, q_0, \delta, \{q_{2n}\})$ where $\delta(q_{k-1}, y[k]) = q_k$ for every $1 \leq k \leq n - 1$, $\delta(q_{n-1}, y[n]) = q_0$, $\delta(q_0, \$) = q_n$, $\delta(q_{n-1+k}, y[k]) = q_{n+k}$ for every $1 \leq k \leq n$, and $\delta(q_{2n}, y[1]) = q_{n+1}$.
- Otherwise $m \geq 1$, D is defined as the tuple $(\Sigma, \{q_0, \ldots, q_{2n+m}\}, q_0, \delta, \{q_{m+2n}\})$ where $\delta(q_{k-1}, x[k]) = q_k$ for every $1 \leq k \leq m$, $\delta(q_{m-1+k}, y[k]) = q_{m+k}$ for every $1 \leq k \leq n - 1$, $\delta(q_{m+n-1}, y[n]) = q_m$, $\delta(q_m, \$) = q_{m+n}$, $\delta(q_{m+n+k-1}, y[k]) = q_{m+n+k}$ for every $1 \leq k \leq n$, and $\delta(q_{m+2n}, y[1]) = q_{m+n+1}$.

One can verify that $L(D) = xy^*\$y^+$ holds and the number of states of D is at most $m + 2n + 1$, i.e., $|x| + 2|y| + 1$.

Proposition 2. *Let $\mathcal{D}_{u\$v}$ be the DFA constructed from the decomposition (u, v) of the ω-word uv^ω. Then $L(\mathcal{D}_{u\$v}) = \{u'\$v' \mid u' \in \Sigma^*, v' \in \Sigma^+, u'v'^\omega = uv^\omega\}$.*

Proof.

- $\subseteq$. This direction is easy since $L(\mathcal{D}_{u\$v}) = \bigcup_{i=1}^{n} L(D_i)$. We only need to prove that for any $1 \le i \le n$, $u'v'^{\omega} = uv^{\omega}$ holds if $u'\$v' \in L(D_i)$. We assume that $L(D_i) = x_i y_i^* \$ y_i^+$. Thus for any $u'\$v' \in L(D_i)$, we have $u' = x_i y_i^j$ and $v' = y_i^k$ for some $j \ge 0, k \ge 1$. It follows that $u'v'^{\omega} = uv^{\omega}$ since $x_i y_i^{\omega} = uv^{\omega}$.
- $\supseteq$. We assume that uv^{ω} is given by the decomposition (u, v). According to Lemma 12, we can get the shortest form of (u, v), say (x, y). Let $n = |y|$. Therefore, $\mathcal{D}_{u\$v}$ is the DFA such that $L(\mathcal{D}_{u\$v}) = \bigcup_{i=1}^{n} L(D_i)$ and $L(D_i) = x_i y_i^* \$ y_i^+$ for each $i \in [1 \cdots n]$ where $x_i = x \cdot y[1 \cdots i]$ and $y_i = y[i + 1 \cdots n] \cdot y[1 \cdots i]$. Let (u', v') be any decomposition of uv^{ω}. We can get the shortest form of (u', v'), say (x_k, y_k) where $x_k = x \cdot y[1 \cdots k]$ and $y_k = y[k+1 \cdots n] \cdot y[1 \cdots k]$. In other words, $u'\$v'$ is accepted by D_k, which indicates that $u'\$v' \in L(\mathcal{D}_{u\$v})$.

Therefore, we complete the proof. ■

Proposition 3. *Let w be an ω-word given by the decomposition (u, v). Then the number of states of $\mathcal{D}_{u\$v}$ constructed from w is in $\mathcal{O}(|v|(|u| + |v|)$.*

The number of states of the DFA D_i in the construction is at most $|u| + 2|r| + 2$ where r is the smallest period of (u, v). Moreover, there are $|r|$ different smallest periods of w. Hence the number of states of $\mathcal{D}_{u\$v}$ is in $\mathcal{O}(|r|(|r| + |u|)) \in \mathcal{O}(|v|(|u| + |v|)$.

8.3. From the FDFA $\mathcal{F}$ to the DFAs $\mathcal{D}_1$ and $\mathcal{D}_2$

In this section, we show how to construct the DFAs $\mathcal{D}_1$ and $\mathcal{D}_2$ from a given FDFA $\mathcal{F} = (M, \{A^u\})$. For each progress DFA $A^u = (\Sigma, Q_u, s_u, F_u, \delta_u)$ of $\mathcal{F}$, we define a DFA $(A^u)^c = (\Sigma, Q_u, s_u, \delta_u, Q_u \setminus F_u)$ such that $L((A^u)^c) = \Sigma^* \setminus L(A^u)$. Note that here δ^u is complete in the sense that $\delta^u(s, a) \neq \emptyset$ for every $s \in Q^u, a \in \Sigma$. In order to construct the DFAs $\mathcal{D}_1$ and $\mathcal{D}_2$, we define two DFAs N_u and $(N_u)^c$ for each state u of M. For the DFA $\mathcal{D}_1$, we define $N_u = M_u^u \times A^u$. Hence $L(N_u) = \{v \in \Sigma^* \mid uv \backsim_M u, v \in L(A^u)\}$. While for the DFA $\mathcal{D}_2$, we define $(N_u)^c = M_u^u \times (A^u)^c$. Similarly, $L((N_u)^c) = \{v \in \Sigma^* \mid uv \backsim_M u, v \notin L(A^u)\}$. Recall that M_u^u is obtained from M by setting the initial state and the accepting state to u. Formally, the DFA construction is defined as follows.

Definition 12. Let $\mathcal{F} = \{M, \{A^u\}\}$ be an FDFA where $M = (\Sigma, Q, q_0, \delta)$. Let N_u (respectively $(N_u)^c$) be defined as the tuple $(\Sigma, Q_u, s_u, \delta_u, F_u)$ for each $u \in Q$. The DFA $\mathcal{D}_1$ (respectively $\mathcal{D}_2$) is defined as the tuple $(\Sigma \cup \{\$\}, Q \cup Q_{Acc}, q_0, \delta \cup \delta_{Acc} \cup \delta_{\$}, F)$ where

$$Q_{Acc} = \bigcup_{u \in Q} Q_u,\ F = \bigcup_{u \in Q} F_u,\ \delta_{Acc} = \bigcup_{u \in Q} \delta_u, \text{ and } \delta_{\$} = \{(u, \$, s_u) \mid u \in Q\}.$$

Intuitively, we use the \$ transitions to connect the leading DFA M and the DFAs N_u (respectively $(N_u)^c$). Figure 15 depicts the DFAs $\mathcal{D}_1$ and $\mathcal{D}_2$ constructed from $\mathcal{F}$ of Fig. 1.

Proposition 4. *Let $\mathcal{F} = (M, \{A^u\})$ be an FDFA and $\mathcal{D}_1$ be the DFA constructed from $\mathcal{F}$ according to Definition 12. Then $L(\mathcal{D}_1) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \backsim_M u, v \in L(A^{M(u)})\}$.*

Proof. According to Definition 12, it is easy to show that for any $u \in \Sigma^*$, $M(u) = \mathcal{D}_1(u)$. Moreover, for any $u, v \in \Sigma^*$, we have that $N_q(v) = \mathcal{D}_1(u\$v)$ since $\mathcal{D}_1$ is a DFA where $q = M(u)$. It follows that we only need to prove that $N_q(v)$ is an accepting state of $\mathcal{D}_1$ if and only if (u, v) is accepted by $\mathcal{F}$.

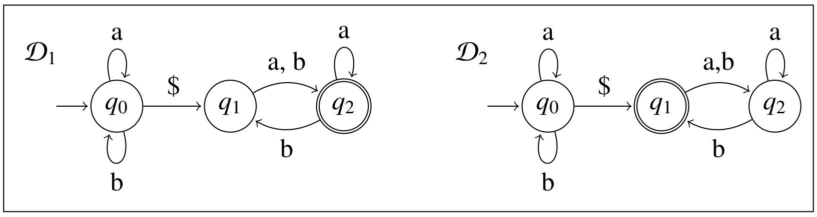


Figure 15: $\mathcal{D}_1$ and $\mathcal{D}_2$ for $\mathcal{F}$ in Fig. 1

- $\Leftarrow$. We prove that if (u, v) is accepted by $\mathcal{F}$, then $u\$v \in L(\mathcal{D}_1)$. Since (u, v) is accepted by $\mathcal{F}$, i.e., $uv \backsim_M u$ and $v \in L(A^q)$, we have that $v \in L(N_q)$ where $q = M(u)$. It follows that $N_q(v)$ is an accepting state. Therefore, $u\$v \in L(\mathcal{D}_1)$.

- $\Rightarrow$. First, we have that $L(\mathcal{D}_1) \subseteq \Sigma^*\$\Sigma^*$ according to Definition 12. For any $u, v \in \Sigma^*$, if $u\$v \in L(\mathcal{D}_1)$, then $\mathcal{D}_1(u\$v)$ is an accepting state. It follows that $v \in L(N_q)$ where $q = M(u)$. Since $N_q = M_q^q \times A^q$, we have that $v \in L(M_q^q)$ and $v \in L(A^q)$, which implies that $uv \backsim_M u$ and $v \in L(A^q)$. Thus, we conclude that (u, v) is accepted by $\mathcal{F}$.

■

Proposition 5. *Let $\mathcal{F} = (M, \{A^u\})$ be an FDFA and $\mathcal{D}_2$ be the DFA constructed from $\mathcal{F}$ according to Definition 12. Then $L(\mathcal{D}_2) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \backsim_M u, v \notin L(A^{M(u)})\}$.*

Proof. Similar to the proof of Proposition 4, for any $u \in \Sigma^*$, we have $M(u) = \mathcal{D}_2(u)$. Moreover, for any $u, v \in \Sigma^*$, we have that $(N_q)^c(v) = \mathcal{D}_2(u\$v)$ since $\mathcal{D}_2$ is a DFA where $q = M(u)$.

- $\Leftarrow$. Assume that $uv \backsim_M u$ and $v \notin L(A^q)$ where $q = M(u)$. On the one hand, $v \in L(M_q^q)$ since $uv \backsim_M u$. On the other hand, $v \in L((A^q)^c)$ since $v \notin L(A^q)$. Therefore, $v \in L((N_q)^c)$ since $(N_q)^c = M_q^q \times (A^q)^c$. It follows that $(N_q)^c(v) = \mathcal{D}_2(u\$v)$ is an accepting state. Therefore, $u\$v \in L(\mathcal{D}_2)$.

- $\Rightarrow$. First, we have that $L(\mathcal{D}_2) \subseteq \Sigma^*\$\Sigma^*$ according to Definition 12. For any $u, v \in \Sigma^*$, if $u\$v \in L(\mathcal{D}_2)$, then $\mathcal{D}_2(u\$v)$ is an accepting state. It follows that $v \in L((N_q)^c)$ where $q = M(u)$. Since $(N_q)^c = M_q^q \times (A^q)^c$, we have that $v \in L(M_q^q)$ and $v \in L((A^q)^c)$, which implies that $uv \backsim_M u$ and $v \notin L(A^q)$.

Proposition 6. *Let n be the number of states of M and k be the number of states in the largest progress DFA of $\mathcal{F}$. The number of states in $\mathcal{D}_1$ (respectively $\mathcal{D}_2$) is in $O(n + n^2k)$.*

Proof. It directly follows from Definition 12. ■

9. Correctness and Complexity Analysis

In this section, we first discuss the correctness of the tree-based FDFA learning algorithm in Sect 9.1 and then present the complexity of the algorithm in Sect. 9.2. Together with the correctness of the BA construction and counterexample analysis, it follows our main result, i.e., Theorem 4 in Sect. 9.2.

9.1. Correctness of the Tree-based FDFA Learning Algorithm

In this section, we let L be the target ω-regular language and we fix an FDFA $\mathcal{F} = (M, \{A^u\})$ and their corresponding classification tree structures $(\mathcal{T}, \{\mathcal{T}_u\})$. Lemma 13 establishes the correctness of our tree-based learning algorithm for the periodic progress trees and the leading tree.

Lemma 13. *Let L be the target language. The tree-based learning algorithm for L will never classify two finite words that belong to the same equivalence class into two different terminal nodes in the leading tree and in the periodic progress trees.*

Proof. We prove by contradiction. Suppose there are two finite words $x_1, x_2 \in \Sigma^*$ that are in the same equivalence class but they are currently classified into different terminal nodes in a classification tree γ.

- $\gamma = \mathcal{T}$ is the leading tree. By assumption, we have the premise $x_1 \backsim_L x_2$. Suppose x_1 and x_2 have currently been assigned to two different terminal nodes t_1 and t_2. Thus we can find the least common ancestor n of t_1 and t_2 from $\mathcal{T}$ with $L_n(n) = (y, v)$ being an experiment to distinguish x_1 and x_2. Assume without loss of generality that t_1 and t_2 are in the left and the right subtrees of n respectively. Therefore, $\mathbf{TE}(x_1, (y, v)) = \mathsf{F}$ and $\mathbf{TE}(x_2, (y, v)) = \mathsf{T}$. It follows that $x_1(yv)^\omega \notin \mathrm{UP}(L)$ and $x_2(yv)^\omega \in \mathrm{UP}(L)$, which implies that $x_1 \not\backsim_L x_2$. Contradiction.
- $\gamma = \mathcal{T}_u$ is a progress tree for an intermediate periodic FDFA. Similarly, we have the premise $x_1 \approx^u_P x_2$. We assume that x_1 and x_2 have been assigned to two different terminal nodes t_1 and t_2 of $\mathcal{T}_u$. Similarly, we find the least common ancestor n of t_1 and t_2 from $\mathcal{T}_u$ with $L_n(n) = v$ being an experiment to distinguish x_1 and x_2. Without loss of generality, we assume that t_1 and t_2 are in the left and the right subtrees of n respectively. Therefore, $\mathbf{TE}(x_1, v) = \mathsf{F}$ and $\mathbf{TE}(x_2, v) = \mathsf{T}$. It follows that $u(x_1v)^\omega \notin \mathrm{UP}(L)$ and $u(x_2v)^\omega \in \mathrm{UP}(L)$, which implies that $x_1 \not\approx^u_P x_2$. Contradiction. ■

We now move onto the learning algorithm for the syntactic FDFA and the recurrent FDFA of L. Recall that in Definition 4 and 5, $x \approx^u_S y$ and $x \approx^u_R y$ are defined according to the right congruence $\backsim_L$. In other words, the syntactic progress trees and the recurrent progress tress are learned according to current leading DFA M. Therefore, the progress trees can be correctly constructed if $\backsim_M$ is already consistent with $\backsim_L$. Recall in the proof of Proposition 1 that $\backsim_M$ is consistent with $\backsim_L$ if and only if for any $x_1, x_2 \in \Sigma^*$, $x_1 \backsim_M x_2 \Longleftrightarrow x_1 \backsim_L x_2$. Lemma 14 gives an important property for the correctness of the tree-based learning algorithm for the syntactic and the recurrent FDFAs.

Lemma 14. *Assume that the learner is learning the target language L. The tree-based algorithm will never classify two finite words $x_1, x_2 \in \Sigma^*$ that belong to the same equivalence class into two different terminal nodes in the syntactic progress trees and the recurrent progress trees if $\backsim_M$ is consistent with $\backsim_L$.*

Proof. Note that the current progress trees in the syntactic and the recurrent FDFAs are constructed with respect to the current leading DFA M in the learning procedure. Let $\mathcal{T}_u$ be the progress tree for a state u of M. We prove the lemma by contradiction.

- $\mathcal{T}_u$ is a progress tree for an intermediate syntactic FDFA. By assumption, we have $x_1 \approx_S^u x_2$. Assume that x_1 and x_2 have been assigned to two different terminal nodes t_1 and t_2 of $\mathcal{T}_u$ respectively. Therefore, we can find the least common ancestor n of t_1 and t_2 from $\mathcal{T}_u$ with $L_n(n) = v$ being an experiment to distinguish x_1 and x_2. According to the definition of **TE** in the syntactic FDFA defined in Sect. 6.2, we let $d_1 := \mathbf{TE}(x_1, v) = (M(ux_1), m_1)$ and $d_2 := \mathbf{TE}(x_2, v) = (M(ux_2), m_2)$ where $m_1, m_2 \in \{A, B, C\}$. $d_1 \neq d_2$ since t_1 and t_2 are in different subtrees of n. It follows that $M(ux_1) \neq M(ux_2)$ or $m_1 \neq m_2$. We show $d_1 = d_2$ by the assumption that M is consistent with $\backsim_L$ in the following cases.

  - Assume that $M(ux_1) \neq M(ux_2)$. According to the definition of $\approx_S^u$ in Definition 4, $ux_1 \backsim_L ux_2$ since $x_1 \approx_S^u x_2$. It immediately follows that $M(ux_1) = M(ux_2)$ since $\backsim_M$ is consistent with $\backsim_L$. Contradiction.
  - Assume that $m_1 \neq m_2$. Similarly, $ux_1 \backsim_L ux_2$ since $x_1 \approx_S^u x_2$. It follows that $M(ux_1) = M(ux_2)$ since $\backsim_M$ is consistent with $\backsim_L$. Moreover, we have that $M(ux_1v) = M(ux_2v)$ for any $v \in \Sigma^*$ since M is a DFA. We discuss the equality of m_1 and m_2 for an experiment $v \in \Sigma^*$ in the following two cases.

    (i) Assume that $u = M(ux_1v)$. Therefore $ux_1v \backsim_L u$ since $\backsim_M$ is consistent with $\backsim_L$. It immediately follows that $u(x_1v)^\omega \in \mathrm{UP}(L) \Longleftrightarrow u(x_2v)^\omega \in \mathrm{UP}(L)$ according to the fact that $x_1 \approx_S^u x_2$. Moreover, we have $u = M(ux_2v)$ since $ux_1 \backsim_L ux_2$. Therefore, it follows that $m_1, m_2 \in \{A, B\}$ according to the definition of **TE** in Sect. 6.2. W.l.o.g., we let $m_1 = A$ and $m_2 = B$, which implies that $u(x_1v)^\omega \in \mathrm{UP}(L)$ while $u(x_2v)^\omega \notin \mathrm{UP}(L)$. Contradiction.

    (ii) Assume that $u \neq M(ux_1v)$. According to the definition of **TE** in Sect. 6.2, we have $m_1 = m_2 = C$. It follows that $d_1 = d_2$ since $M(ux_1) = M(ux_2)$. Contradiction.

  Therefore, t_1 and t_2 will be the same terminal node if $\backsim_M$ is consistent with $\backsim_L$.

- $\mathcal{T}_u$ is a progress tree for an intermediate recurrent FDFA. This proof is analogous to the proof for the syntactic FDFA. We first have premise $x_1 \approx_R^u x_2$. Assume that x_1 and x_2 have been assigned to two different terminal nodes t_1 and t_2 of $\mathcal{T}_u$ respectively. Thus we can find the least common ancestor n of t_1 and t_2 from $\mathcal{T}_u$ with $L_n(n) = v$ being an experiment to distinguish x_1 and x_2. Let $d_1 := \mathbf{TE}(x_1, v)$ and $d_2 := \mathbf{TE}(x_2, v)$ where $d_1, d_2 \in \{\mathsf{F}, \mathsf{T}\}$. $d_1 \neq d_2$ since t_1 and t_2 are in different subtrees of n. W.l.o.g., we let $d_1 = \mathsf{F}$ and $d_2 = \mathsf{T}$. According to the definition of **TE** in Sect. 6.2, we have $u = M(ux_2v)$ and $u(x_2v)^\omega \in \mathrm{UP}(L)$ since $d_2 = \mathsf{T}$. It follows that $ux_2v \backsim_L u$ since $\backsim_M$ is consistent with $\backsim_L$. Moreover, we have that $u = M(ux_1v)$ and $u(x_1v)^\omega \in \mathrm{UP}(L)$ since $x_1 \approx_R^u x_2$. According to the definition of **TE**, we have $d_1 = \mathsf{T}$. Contradiction.

Consequently, we can conclude that the classification of equivalence classes in the progress trees will be correct if $\backsim_M$ is consistent with $\backsim_L$. ■

We have proved that the tree-based learning algorithm will not do any “bad” things, i.e., it will not classify any two different words that belong to the same equivalence class into different terminal nodes if $\backsim_M$ is currently consistent with $\backsim_L$. Recall in Lemma 4 that the tree-based algorithm will do some “good” things whenever receiving a counterexample (u, v) from the FDFA teacher, i.e., there will be a new state added to either the leading DFA M or to a progress DFA $A^{M(u)}$ after each refinement step. In other words, Lemma 4 guarantees that the learning algorithm will either make progress for the leading DFA or the corresponding progress DFA after every refinement.

However, the learning algorithm may make progress for the progress DFAs in a wrong direction when $\backsim_M$ is not consistent with $\backsim_L$. More precisely, according to Lemma 14, it happens that in the progress syntactic trees and the recurrent progress trees, two finite words that belong to the same equivalence class can be classified into different terminal nodes if $\backsim_M$ is not consistent with $\backsim_L$. One worry is that if the FDFA teacher always chooses to refine the progress DFAs when $\backsim_M$ is not consistent with $\backsim_L$, the learning algorithm may not terminate. Lemma 15 shows that the learning algorithm will terminate since the number of equivalence classes of the progress DFAs with respect to current leading DFA M is finite. More precisely, if we fix a leading DFA M and a state u of M, we are actually learning a DFA defined by a right congruence $\approx^u_{S'}$ when learning the progress DFA A^u of the target syntactic FDFA of L. We define $x \approx^u_{S'} y$ if and only if $M(ux) = M(uy)$ and for any $v \in \Sigma^*$, if $M(uxv) = u$, then $u(xv)^\omega \in L \Longleftrightarrow u(yv)^\omega \in L$. One can easily verify that $x \approx^u_{S'} y$ is a right congruence and that if $\backsim_M$ is consistent with $\backsim_L$, then $x \approx^u_{S'} y$ is equivalent to $x \approx^u_S y$.

Lemma 15. *Let M be a leading DFA with a set of states Q. For every state u of M, the index of $\approx^u_{S'}$ is bounded by $|Q| \cdot | \approx^u_P |$.*

Proof. Recall that we use a word to represent a state in M. Here u is a representative word of the state it represents. Let $x \in \Sigma^*$. We show how to classify x into an equivalence class of $\approx^u_{S'}$ in the following. According to the definition of $\approx^u_{S'}$, we first find the state $q = M(ux)$. Note that for every $y \in \Sigma^*$ such that $q \neq M(uy)$, x and y must not belong to the same equivalence class. Therefore, x can be first classified into one of $|Q|$ classes since the number of possible choices of q is $|Q|$. Further, in order to distinguish x from a finite word y such that $q = M(uy)$, we have to check whether for $\forall v \in \Sigma^*$, if $M(uxv) = u$, $u(xv)^\omega \in L \Longleftrightarrow u(yv)^\omega \in L$ holds, i.e., whether $x \approx^u_P y$ holds. Thus we can use $(q, [x]_{\approx^u_P}) \in Q \times \Sigma^*/_{\approx^u_P}$ as a label to represent the equivalence class of $\Sigma^*/_{\approx^u_{S'}}$ which x belongs to. Therefore, it follows that the index of the right congruence $\approx^u_{S'}$ is at most $|Q| \cdot | \approx^u_P |$. ■

Similarly, we also have a right congruence $\approx^u_{R'}$ for the recurrent FDFA learning. We define $x \approx^u_{R'} y$ if and only if for every $v \in \Sigma^*$, $M(uxv) = u \wedge u(xv)^\omega \in L \Longleftrightarrow M(uyv) = u \wedge u(yv)^\omega \in L$. Since $x \approx^u_{S'} y$ implies $x \approx^u_{R'} y$, it follows that $| \approx^u_{R'} |$ is smaller than $| \approx^u_{S'} |$. The implication from $x \approx^u_{S'} y$ to $x \approx^u_{R'} y$ can be easily established as follows: firstly, we have that $u(yv)^\omega \in L$ by the assumption that $uxv \backsim_M u \wedge u(xv)^\omega \in L$ and $x \approx^u_{S'} y$; secondly, together with the assumption $uxv \backsim_M u$ and the fact that $ux \backsim_M uy$ holds since $x \approx^u_{S'} y$, we conclude that $uyv \backsim_M u$ since M is a DFA. Therefore, a similar result for the recurrent FDFA follows in Lemma 16.

Lemma 16. *Let M be a leading DFA with a set of states Q. For every state u of M, the index of $\approx^u_{R'}$ is bounded by $|Q| \cdot | \approx^u_P |$.*

Theorem 3. *The tree-based FDFA learning algorithm will terminate and can learn the three canonical FDFAs if there is an FDFA teacher answering membership and equivalence queries about the target canonical FDFA of L.*

Proof (Sketch). The tree-based learning algorithm for the periodic FDFA will terminate and learn an FDFA $\mathcal{F}$ such that $\mathrm{UP}(\mathcal{F}) = \mathrm{UP}(L)$ since the number of states in $\mathcal{F}$ is finite. This result can be justified by Lemmas 13 and 4.

According to Lemma 13 and Lemma 14, the tree-based algorithm for the syntactic and the recurrent FDFAs can classify finite words correctly if $\backsim_M$ is consistent with $\backsim_L$. If $\backsim_M$ is not consistent with $\backsim_L$, the FDFA teacher will be able to return a counterexample to refine current M. If the leading DFA has been refined, the algorithm for the syntactic and the recurrent FDFAs will learn all progress DFAs from scratch with respect to the new leading DFA M. The learning procedure for the progress DFAs will terminate, which is justified by Lemmas 4, 15 and 16. At some point, $\backsim_M$ will be consistent with $\backsim_L$ since the number of states in the conjectured FDFA increases after every refinement. Thus, the learning algorithm will terminate since the numbers of states in the corresponding syntactic and recurrent FDFAs of L are both finite. Therefore, we complete the proof. ■

9.2. Complexity of the Tree-based Learning Algorithm

In this section, we denote by $\mathcal{F} = (M, \{A^u\})$ the periodic FDFA of the target ω-regular language L. Unless stated otherwise, we let n be the number of states in the leading DFA M and k be the number of states in the largest progress DFA A^u. We remark that $\mathcal{F}$ is uniquely defined for L according to Definition 3 and the table-based algorithm needs the same amount of equivalence queries as the tree-based one in the worst case. We define the length of a decomposition (u, v) as the sum of the lengths of u and v, i.e., $|(u, v)| = |u| + |v|$. In the following, we first discuss the complexity of the operations of the FDFA teacher in Proposition 7 and then consider the complexity of the operations of the FDFA learner in Theorems 4 and 5. At last, we will present the main result of this paper in Theorem 6.

Proposition 7. *Let $\mathcal{F} = (M, \{A^u\})$ be the current conjectured FDFA. Assume that the counterexample uv^ω returned by the BA teacher is given by the decomposition (u, v). Then*

- *the time and space complexity for building the BAs $\underline{B}, \overline{B}$ and the LDBAs $\underline{B}_d$ and $\overline{B}_d$ are in $O(n^2k^3), O(n^2k^2), O(n^3k^5)$ and $O(n^3k^3)$ respectively, and*
- *for the under approximation method, the time and space complexity for analyzing the counterexample uv^ω are in $O(n^2k \cdot (|v|(|v| + |u|))$, while for the over approximation method, the time and space complexity for analyzing uv^ω are in $O(n^2k^2 \cdot (|v|(|v| + |u|))$ and in $O(n^2k(|v|(|v| + |u|))$ respectively.*

Proof.

- This is an immediate result of Lemma 6 and Corollary 1.
- According to Propositions 3 and 6, the numbers of states in the DFAs $\mathcal{D}_1$ and $\mathcal{D}_2$ are both in $O(n + n^2k)$ and the number of states in $\mathcal{D}_{u\$v}$ is at most $|v|(|v| + |u|)$ since (u, v) is the returned decomposition of the counterexample uv^ω. Note that except for the case O3 in the over-approximation method, $\mathcal{D}_{u\$v}$, $\mathcal{D}_1$ and $\mathcal{D}_2$ are used in counterexample analysis. Thus, except for the case O3, the time and space complexity for the counterexample analysis are both in $O(n^2k \cdot (|v|(|v| + |u|)))$. When we analyze the spurious negative counterexample, i.e., the case O3, the time and space complexity are in $O(nk(n + nk) \cdot (|v|(|v| + |u|)))$ and $O((n + nk) \cdot (|v|(|v| + |u|)))$ respectively according to Lemma 7. Since the time and space complexity for case O1 and the case O2 are both in $O(n^2k \cdot (|v|(|v| + |u|))$, it follows that the time and space complexity for the over-approximation method are in $O(n^2k^2 \cdot (|v|(|v| + |u|))$ and in $O(n^2k(|v|(|v| + |u|))$ respectively.

Therefore, we complete the proof. ■

Theorem 4 (Query Complexity). *Let (u, v) be the longest counterexample returned from the FDFA teacher. The number of equivalence queries needed for the tree-based FDFA learning algorithm to learn the periodic FDFA of L is in $O(n + nk)$, while the number of membership queries is in $O((n + nk) \cdot (|u| + |v| + (n + k) \cdot |\Sigma|))$.*

For both the syntactic and the recurrent FDFAs, the number of equivalence queries needed for the tree-based FDFA learning algorithm is in $O(n + n^3k)$, while the number of membership queries is in $O((n + n^3k) \cdot (|u| + |v| + (n + nk) \cdot |\Sigma|))$.

Proof. Theorem 4 can be justified according to Lemma 13, Lemma 15, Lemma 16 and Theorem 3. Recall that we let $\mathcal{F} = (M, \{A^u\})$ be the unique periodic FDFA of L. The number of states of M is n and k is the number of the largest progress DFA of $\mathcal{F}$.

Let (u, v) be a counterexample from the FDFA teacher. The number of membership queries needed for the FDFA learner to find the right experiment to refine the leading DFA is at most $|u|$ and that for refining the corresponding progress DFA is at most $|v|$. Therefore, the number of membership queries used in finding the right experiment for the FDFA learner is bounded by $|u| + |v|$. We remark that one can also reduce the number of membership queries used by the FDFA learner to $\log(|u| + |v|)$ since one can use a binary search instead of the linear search in Sect. 6.2 to find a suffix of u or v as an experiment for refinement. Basically, the linear search in Sect. 6.2 tries to find a position in u (respectively v) where the result of the experiment function **TE** differs from that of the previous position. Therefore, by repeatedly dividing the sequence of positions in half such that the results of the first position and the last position still differ, we only need $\log(|u| + |v|)$ queries to find the right position.

Membership queries are also needed in constructing the corresponding DFA after a classification tree has been refined. Assume that the new added terminal node is labeled by p, the terminal node to be refined is labeled by q and the experiment is e. We only need to ask membership queries to compute the successors of p and update the successors of the predecessors of q as follows. (i) Computing the successors of p needs to calculate $\delta(p, a)$ for every $a \in \Sigma$, which requires $|\Sigma| \cdot h$ membership queries where h is the height of the classification tree. (ii) Updating the successors of the predecessors of q needs to calculate $\mathbf{TE}(s, e)$ for every state s and $a \in \Sigma$ for which currently we have $\delta(s, a) = q$; this requires at most $|\Sigma| \cdot m$ membership queries where m is the number of states in M or $A^{M(u)}$. Since the height of the classification tree is at most m, the number of membership queries needed for constructing the new conjectured DFA is at most $2 \cdot m \cdot |\Sigma|$. It follows that for the tree-based algorithm, the number of membership queries used in the counterexample guided refinement is bounded by $|u| + |v| + 2m \cdot |\Sigma|$. We remark that in the table-based algorithm, the number of membership queries used in the counterexample guided refinement is bounded by $|u| + |v| + m + |\Sigma| \cdot m + |\Sigma|$, where $|u| + |v|$ membership queries are used for finding the experiment and $m + (m + 1)|\Sigma|$ membership queries are used to fill the table.

The tree-based algorithm for the three canonical FDFAs are as follows.

During the learning procedure for the periodic FDFA, when receiving a counterexample for the FDFA learner, the tree-based algorithm either adds a new state into the leading DFA or into a progress DFA. Thus, the number of equivalence queries is bounded by $n + nk$ since the number of states in the target periodic FDFA of L is at most $n + nk$. Note that $m \leq n + k$ since we will either refine the leading DFA or a progress DFA whenever receiving a counterexample. Therefore, the number of membership queries needed for learning the periodic FDFA is bounded by $(n + nk) \cdot (|u| + |v| + 2(n + k) \cdot |\Sigma|) \in O((n + nk) \cdot (|u| + |v| + (n + k) \cdot |\Sigma|))$ in the worst case.

Now we consider the learning algorithm for the syntactic and the recurrent FDFAs. On receiving a counterexample for the FDFA learner, the tree-based algorithm will first decide whether to refine the leading DFA or a progress DFA. If it decides to refine the leading DFA, we need to initialize all progress trees as a single node labeled by ϵ again. Thus the number of states in the progress DFAs of the conjectured FDFA may decrease at that point. Otherwise it decides to refine a progress DFA and the number of states of the conjectured FDFA will be increased by one.

In the worst case, the FDFA learner will try to learn the progress DFAs as much as possible. Therefore, if the current leading DFA has m states, the learner will learn a DFA is of size at most $m \cdot k$ according to Lemmas 15 and 16 for every progress DFA. Once all progress trees cannot be refined any more, either the learning task finishes or the FDFA teacher returns a counterexample to refine the leading DFA. For the latter case, the number of states in the leading DFA will increase by one, i.e., $m + 1$, and the learner has to redo the learning work for all progress trees from scratch. The number of states of all progress DFAs in the new conjectured FDFA is bounded by $(m + 1)^2 \cdot k$. Assume that the learner will keep learning the target FDFA in this way. Then the number of equivalence queries needed for the tree-based algorithm is bounded by $(1 + 1 \cdot 1 \cdot k) + (1 + 2 \cdot 2 \cdot k) + \cdots (1 + (n - 1) \cdot (n - 1) \cdot k) + (1 + n \cdot n \cdot k) \in O(n + n^3 k)$. Similarly, in the syntactic and the recurrent FDFAs, we have that $m \leq n + nk$ since the number of states in a progress DFA is bounded by nk. It follows that the number of membership queries needed for the algorithm is in $O((n + n^3 k) \cdot (|u| + |v| + 2(n + nk) \cdot |\Sigma|)) \in O((n + n^3 k) \cdot (|u| + |v| + (n + nk) \cdot |\Sigma|))$ in the worst case. ■

Theorem 5 (Space Complexity). *Let n be the number of states in the leading DFA M and k be the number of states in the largest progress DFA. For the tree-based learning algorithm of the three canonical FDFAs, the space required to learn the leading DFA is in $O(n)$. For the periodic FDFA, the space required for the tree-based algorithm to learn each progress DFA is in $O(k)$, while for the syntactic and the recurrent FDFAs, the space required is in $O(nk)$. For the table-based learning algorithm of the three canonical FDFAs, the space required to learn the leading DFA is in $O((n + n \cdot |\Sigma|) \cdot n)$. For the periodic FDFA, the space required for the table-based algorithm to learn each progress DFA is in $O((k + k \cdot |\Sigma|) \cdot k)$, while for the syntactic and the recurrent FDFAs, the space required is in $O((nk + nk \cdot |\Sigma|) \cdot nk)$.*

Proof. As we mentioned in Sect. 5, the FDFA learner can be viewed as a learner consisting of many component DFA learners. Assume that the number of the states in the target DFA is m for a component DFA learner. For the table-based component DFA learner, the size of the observation table is in $O((m + m \cdot |\Sigma|) \cdot m)$ since there are $m + m \cdot |\Sigma|$ rows and at most m columns in the observation table in the worst case. In contrast, for the tree-based component DFA learner, the number of nodes in the classification tree is in $O(m)$ since the number of terminal nodes in the classification tree is m and the number of internal nodes is at most $m - 1$.

For the periodic FDFA, the number of states in the conjectured FDFA will increase after each refinement step. Thus, it is easy to conclude that the space required for the leading DFA is in $O(n)$ if we use the tree-based learning algorithm and the space required by the table-based algorithm is in $O((n + n \cdot |\Sigma|) \cdot n)$. Similarly, the space required by the tree-based learning algorithm to learn each progress DFA is in $O(k)$, while for the table-based algorithm, the space required is in $O((k + k \cdot |\Sigma|) \cdot k)$.

For the syntactic and the recurrent FDFAs, the component DFA learner for the leading DFA is the same as the one for the periodic FDFA. Thus the space required by the table-based and the tree-based algorithms remains separately the same. For learning the progress DFAs, the number of states in each progress DFA is at most nk according to Lemmas 15 and 16. Therefore, for the table-based algorithm, the space required is in $O((nk + nk \cdot |\Sigma|) \cdot nk)$. While for the tree-based algorithm, the space required to learn each progress DFA is in $O(nk)$. ■

Theorem 6 (Correctness and Termination). *The BA learning algorithm based on the under-approximation method always terminates, and returns a BA accepting the unknown ω-regular language L. If the BA learning algorithm based on the over-approximation method terminates without reporting an error, it returns a BA accepting the ω-regular language L. Our BA learning algorithms run in polynomial time w.r.t. the number of states of the periodic FDFA of L.*

Proof. If the algorithm uses the under-approximation method to construct BAs, the learning algorithm has to first learn a canonical FDFA to get a right conjectured BA in the worst case. If the BA learning algorithm based on the over-approximation method terminates without reporting an error when finding a counterexample for the FDFA learner in case O3, the output BA B must accept L since B has passed the equivalence query. One can verify that all the operations needed for our BA learning algorithm run in polynomial time in the number of states of the periodic FDFA of L. This theorem is justified by Lemma 2, Lemma 6 and Theorem 3. ■

By Theorem 6, our BA learner based on the under-approximation method has to learn a canonical FDFA of L to obtain a BA of L in the worst case. In practice, the learning algorithm very often finds a BA accepting L before converging to a canonical FDFA.

10. Experimental results

All the learning algorithms proposed in this work are implemented in the ROLL library [42] (`http://iscasmc.ios.ac.cn/roll`). In the ROLL library, all DFA operations are delegated to the *dk.brics.automaton* package, and we use the RABIT tool [55, 56] to check the equivalence of two BAs. We evaluate the performance of our learning algorithms using the smallest BAs corresponding to all the 295 LTL specifications available in Büchi Store[43], where the numbers of states in the BAs range from 1 to 17 and the numbers of transitions from 0 to 123. Besides this, we also evaluate our algorithms on 20 BAs whose languages cannot be expressed by LTL formulas where the numbers of states range from 2 to 21 and the numbers of transitions from 3 to 41. The machine we used for the experiments is a 2.5 GHz Intel Core i7-6500 with 4 GB RAM. We set the timeout period to 30 minutes.

Experiments on 295 LTL specifications using conversions from FDFAs to BAs. The overall experimental results for 295 LTL specifications are given in Table 1. In this section, we use $L^{\$}$ to denote the ω-regular learning algorithm in [36], and L^{Periodic}, $L^{\mathsf{Syntactic}}$ and $L^{\mathsf{Recurrent}}$ to represent the periodic, syntactic and recurrent FDFA learning algorithm introduced in Sect. 5 and 6. From the table, we can find the following facts: (1) The BAs learned from $L^{\$}$ have more states but fewer transitions than their FDFA based counterparts. (2) L^{Periodic} uses fewer membership queries compared to $L^{\mathsf{Syntactic}}$ and $L^{\mathsf{Recurrent}}$. The reason is that $L^{\mathsf{Syntactic}}$ and $L^{\mathsf{Recurrent}}$ need to restart the learning of all progress DFAs from scratch when the leading DFA has been modified. (3) Tree-based algorithms always solve more learning tasks than their table-based counterparts. In particular, the tree-based $L^{\mathsf{Syntactic}}$ with the under-approximation method solves all 295 learning tasks.

Table 1: Overall experimental results for 295 BAs of LTL specifications. We show the results of 285 cases for which all algorithms can finish the BA learning within the timeout period and list the number of cases cannot be solved (#Unsolved). The mark n^*/m denotes that there are n cases terminate with an error (in the over-approximation method) and $m-n$ cases run out of time. The rows #St., #Tr., #MQ, and #EQ, are the numbers of states, transitions, membership queries, and equivalence queries. Time$_{eq}$ is the time spent in answering equivalence queries and Time$_{total}$ is the total execution time. EQ(%) is the percentage of the time for the equivalence queries in the total running time.

Models	$L^{\$}$		$L^{\textsf{Periodic}}$				$L^{\textsf{Syntactic}}$				$L^{\textsf{Recurrent}}$			
Struct.&	Table	Tree	Table		Tree		Table		Tree		Table		Tree	
Approxi.			under	over	under	over	under	over	under	over	under	over	under	over
#Unsolved	4	2	3	0/2	2	0/1	1	4*/5	0	3*/3	1	0/1	1	0/1
#St.	3078	3078	2481	2468	2526	2417	2591	2591	**2274**	**2274**	2382	2382	2400	2400
#Tr.	10.6k	**10.3k**	13.0k	13.0k	13.4k	12.8k	13.6k	13.6k	12.2k	12.2k	12.7k	12.7k	12.8k	12.8k
#MQ	105k	114k	86k	85k	69k	**67k**	236k	238k	139k	139k	124k	124k	126k	126k
#EQ	**1281**	2024	1382	1351	1950	1918	1399	1394	2805	2786	1430	1421	3037	3037
Time$_{eq}$(s)	146	817	580	92	186	159	111	115	**89**	91	149	149	462	465
Time$_{total}$(s)	183	861	610	114	213	186	140	144	**118**	120	175	176	499	501
EQ(%)	79.8	94.9	95.1	80.7	87.3	85.5	79.3	79.9	**75.4**	75.8	85.1	84.6	92.6	92.8

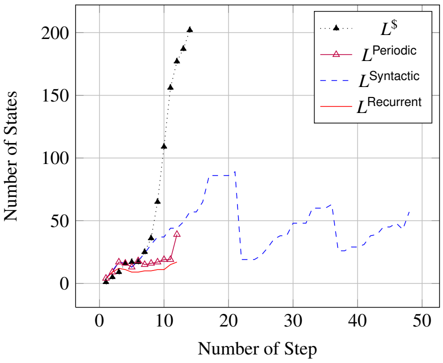


Figure 16: Growth of state counts in BA

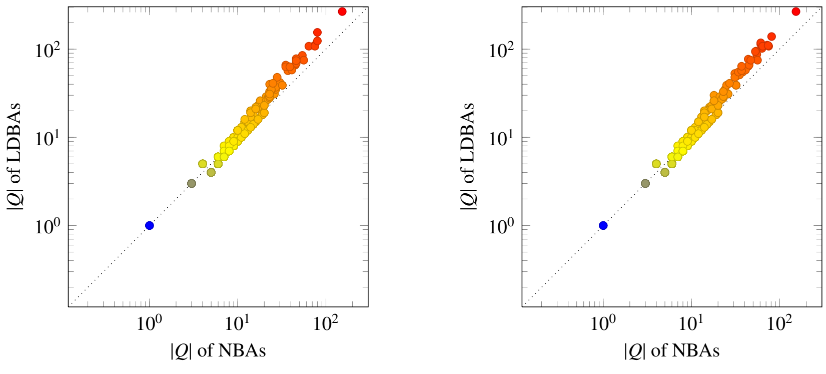
In the experiment, we observe that table-based $L^{\$}$ has 4 cases that cannot be finished within the timeout period, which is the largest number among all learning algorithms[3]. That's because for these 4 cases, the average time required for $L^{\$}$ to get an equivalence query result is much longer than the FDFA algorithms. After a careful examination, we find that the growth rate of the size (number of states) of the conjectured BAs generated by table-based $L^{\$}$ is much faster than that of table-based FDFA learning algorithms. In Fig. 16, we illustrate the growth rate of the size (number of states) of the BAs generated by each table-based learning algorithm using one learning task that cannot be solved by $L^{\$}$ within the timeout period. The figures of the other three learning tasks show the same trend and hence are omitted. Another interesting observation is that the sizes of BAs generated by $L^{\textsf{Syntactic}}$ can decrease in some iterations because the leading DFA is refined at those iterations and thus the algorithms have to redo the learning of all progress automata from scratch.

To our surprise, in our experiment, the size of BAs $\overline{B}$ produced by the overapproximation method is not much smaller than the BAs $\underline{B}$ produced by the underapproximation method. Recall that the automaton $\overline{B}$ comes from the product of three DFAs $M_u^u \times (A^u)_v^{s_u} \times (A^u)_v^v$ while

[3] Most of the unsolved tasks using the over-approximation method are caused by the situation that the FDFA teacher cannot find a valid counterexample for refinement.

the automaton $\underline{B}$ comes from the product of only two DFAs $M_u^u \times (A^u)_v^{s_u}$ (Sect. 7). The reason for the above observations is that very often the language of the product of three DFAs is equivalent to the language of the product of two DFAs, thus we get the same DFA after applying DFA minimizations. We note that the languages of these two products are not necessarily equivalent in theory. Nevertheless, the over-approximation method is still helpful for L^{Periodic} and $L^{\mathsf{Recurrent}}$. For L^{Periodic}, the over-approximation method solves more learning tasks than the under-approximation method. For $L^{\mathsf{Recurrent}}$, the over-approximation method solves one tough learning task that is not solved by the under-approximation method.

As we mentioned at the end of Sect. 6.2, a possible optimization is to reuse the counterexamples and to avoid equivalence queries as much as possible. The optimization helps the learning algorithms to solve 9 more cases that were not solved before.



Over-Approximation

Under-Approximation

Figure 17: Comparison of the number of states between NBAs and LDBAs learned from tree-based algorithms

Experiments on 295 LTL specifications using conversions from FDFAs to LDBAs. The number of states of a LDBA constructed from an FDFA $\mathcal{F}$ by Definition 9 is quadratically larger than that in the corresponding NBA constructed by Definition 8. Therefore, in order to learn a LDBA we can construct NBAs for the equivalence queries and construct a LDBA from the FDFA for final result once the learning task succeeds. We use this strategy to learn the LDBAs from the BAs of the Büchi Store. Since the learning procedure is the same as usual except we construct a LDBA from the final FDFA, we only compare the number of states between NBAs and LDBAs learned from the tree-based algorithms in Fig. 17. The numbers of states of NBAs and LDBAs learned from table-based algorithms share the same trend and thus the comparison is omitted. From Fig. 17, we can see that most of points are above the diagonal, which indicates the number of states in LDBAs are larger than those in NBAs when we consider large BAs. This is because the number of states of LDBAs are quadratically larger than those of NBAs. We also observe that there are a lot of points which are below the diagonal. The reason is that the construction of LDBAs in Definition 9 requires first remove all states that cannot reach the accepting states in the DFA, while the construction of BAs in Definition 8 does not use the same removal operation.

Experiments on 20 BAs using conversions from FDFAs to BAs. It is known that LTL formulas can only express a strict subset of the ω-regular languages [57]. For instance, the language $L_k = \{w \in \{a, b\}^\omega \mid a \text{ appears after every } k \text{ letters in } w\}$ where $k \geq 1$, which can be expressed as

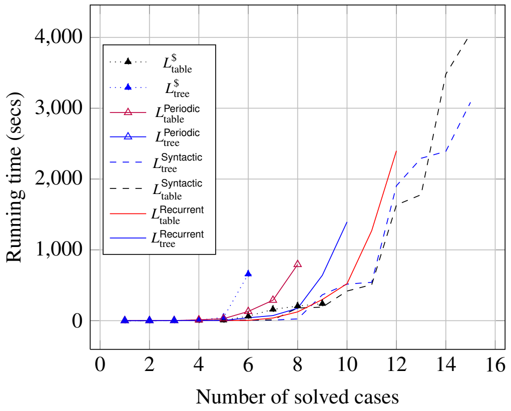


Figure 18: Experiments for 20 BAs whose languages are not recognizable with LTL formulas

the ω-regular expression $((a+b)^k \cdot a)^\omega$ and thus recognized by a BA , but not by any LTL formula. This is because an LTL formula for L_k, if exists, would need to count the number of letters before a, which is clearly beyond the expressive power of LTL. To show that our algorithms are also efficient in learning ω-regular languages that cannot be recognized by LTL formulas, we conducted experiments on 20 BAs B_k accepting $L_k = ((a+b)^k \cdot a)^\omega$ with $1 \leq k \leq 20$; each B_k has $k+1$ states and the results are depicted in Fig. 18 showing how the running time of each learning algorithm evolves with respect to the number of solved cases. Note that we have excluded in Fig. 18 the running time for the cases that cannot be solved within 30 minutes. We note that only the under-approximation method is used here for constructing BAs since learning algorithms based on over-approximation often fail to obtain a valid counterexample for FDFAs due to their incompleteness. We summarize our observations below. Among all learning algorithms, $L^{\text{Syntactic}}_{\text{table}}$ and $L^{\text{Syntactic}}_{\text{tree}}$ complete the most number of learning tasks (15 BAs). Moreover, the set of cases solved by either $L^{\text{Syntactic}}_{\text{table}}$ or $L^{\text{Syntactic}}_{\text{tree}}$ covers all the 17 cases solved by at least one algorithm. In particular, $L^{\text{Syntactic}}_{\text{tree}}$ solves all cases that can be solved by either $L^{\$}$ or $L^{\text{Recurrent}}$ (regardless of table-based or tree-based ones). $L^{\text{Syntactic}}_{\text{tree}}$ can solve 2 cases that $L^{\text{Syntactic}}_{\text{table}}$ cannot manage, and vice versa. We note that L^{Periodic} and $L^{\$}$ both solve less than 10 cases due to their fast growth of states in constructed BAs. It follows that L^{Periodic} and $L^{\$}$ generally perform worse than $L^{\text{Recurrent}}$ and $L^{\text{Syntactic}}$ in terms of the number of solved cases, which is consistent with the results summarized in Table 1. Contrary to the results for 295 LTL specifications in Table 1, from Fig. 18 we can see that table-based algorithms usually solve at least as many cases as their tree-based counterparts. However, we want to point out that the cases solved by tree-based algorithms may not be solved by their table-based counterparts, which indicates that tree-based algorithms are good complement to table-based ones. More significantly, $L^{\text{Syntactic}}_{\text{tree}}$ has the best performance among all algorithms regarding the number of solved cases and the total running time for learning tasks on all benchmarks considered in this work.

11. Discussion and Future works

Regarding our experiments, the BAs used as target automata are in general small; the average size of the input BAs are around 10 states. From our experience of applying DFA learning algorithms, the performance of tree-based algorithms is significantly better than the table-based ones when the number of states of the learned DFA is large, say more than 1000. We believe this will also apply to the case of BA learning. Nevertheless, in our current experiments, most of the time are spent in answering equivalence queries. One possible direction to improve the scale of the experiment is to use a PAC (probably approximately correct) BA teacher [58] instead of an exact one, so the equivalence queries can be answered faster because the BA equivalence queries will be replaced with a bunch of BA membership testing.

There are several avenues for future works. We believe that our learning algorithms are an interesting contribution for the community because they bring the possibility of many applications. For the next step, we will investigate the possibility of applying BA learning to the problem of reactive system synthesis, which is known to be a very difficult problem. Recently, Angluin *et al.* proposed in [59] a polynomial-time learning algorithm for a class T of ω-tree languages derived from weak regular ω-word languages based on the learning algorithm for weak regular ω-word languages by Maler and Pnueli [35]. It is worth exploring whether we can find a way to learn a class of ω-tree languages larger than T based on our learning algorithm for the complete class of regular ω-word languages.

There are learning algorithms for residual NFA [2], which is a more compact canonical representation of regular languages than DFA. We think maybe one can also generalize the learning algorithm for family of DFAs to family of residual NFAs (FRNFA). To do this, one needs to show FRNFAs also recognize ω-regular language and finds the corresponding right congruences. Interestingly, a learning algorithm for finite automata over infinite alphabets is proposed in [60]. A natural direction of future work is to extend our learning algorithm for BAs over finite alphabets to one that supports infinite alphabets. Then one needs to develop a learning algorithm for FDFAs over infinite alphabets and an equivalent translation from FDFAs to BAs.

The automata learning paradigms in literature can be generally classified into *active* and *passive*. The learning algorithms considered in this work are active since they actively interact with a teacher to learn about the target language. In contrast, passive learning algorithms learn an automaton representation for the target language only from a given set of data. It is shown in [61] that a strict class of deterministic BAs can be passively learned using polynomial time and data, while this is not the case for nondeterministic BAs. One possible direction for improving the performance of learning a nondeterministic BA from data is that one first develops a passive learning algorithm for FDFAs and then converts FDFAs to BAs.

Acknowledgments. We thank two anonymous reviewers for their valuable suggestions to improve the presentation of this paper. This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 61761136011,61836005,61532019), the Guangdong Science and Technology Department (Grant No. 2018B010107004), and the MOST project No. 103-2221-E-001-019-MY3.

[1] D. Angluin, D. Fisman, Learning regular omega languages, Theor. Comput. Sci. 650 (2016) 57–72. doi:10.1016/j.tcs.2016.07.031.

[2] B. Bollig, P. Habermehl, C. Kern, M. Leucker, Angluin-style Learning of NFA, in: IJCAI, 2009, pp. 1004–1009.

[3] M. J. Kearns, U. V. Vazirani, An Introduction to Computational Learning Theory, MIT Press, Cambridge, MA, USA, 1994.

[4] D. Angluin, Learning regular sets from queries and counterexamples, Inf. Comput. 75 (2) (1987) 87–106. doi:10.1016/0890-5401(87)90052-6.

[5] R. L. Rivest, R. E. Schapire, Inference of finite automata using homing sequences (extended abstract), in: D. S. Johnson (Ed.), Proceedings of the 21st Annual ACM Symposium on Theory of Computing, ACM, 1989, pp. 411–420. doi:10.1145/73007.73047.

[6] J. M. Cobleigh, D. Giannakopoulou, C. S. Pasareanu, Learning assumptions for compositional verification, in: H. Garavel, J. Hatcliff (Eds.), TACAS, Vol. 2619 of Lecture Notes in Computer Science, Springer, 2003, pp. 331–346. doi:10.1007/3-540-36577-X_24.

[7] S. Chaki, E. M. Clarke, N. Sinha, P. Thati, Automated assume-guarantee reasoning for simulation conformance, in: K. Etessami, S. K. Rajamani (Eds.), CAV, Vol. 3576 of Lecture Notes in Computer Science, Springer, 2005, pp. 534–547. doi:10.1007/11513988_51.

[8] Y. Chen, A. Farzan, E. M. Clarke, Y. Tsay, B. Wang, Learning minimal separating dfa's for compositional verification, in: S. Kowalewski, A. Philippou (Eds.), TACAS, Vol. 5505 of Lecture Notes in Computer Science, Springer, 2009, pp. 31–45. doi:10.1007/978-3-642-00768-2_3.

[9] O. Grumberg, Y. Meller, Learning-based compositional model checking of behavioral UML systems 45 (2016) 117–136. doi:10.3233/978-1-61499-627-9-117.

[10] S.-W. Lin, E. Andre, Y. Liu, J. Sun, J. Dong, Learning assumptions for compositionalverification of timed systems, IEEE Transactions on Software Engineering 40 (2014) 137–153. doi:10.1109/TSE.2013.57.

[11] R. Alur, P. Madhusudan, W. Nam, Symbolic compositional verification by learning assumptions, in: K. Etessami, S. K. Rajamani (Eds.), CAV, Vol. 3576 of Lecture Notes in Computer Science, Springer, 2005, pp. 548–562. doi:10.1007/11513988_52.

[12] L. Feng, M. Z. Kwiatkowska, D. Parker, Automated learning of probabilistic assumptions for compositional reasoning, in: D. Giannakopoulou, F. Orejas (Eds.), FASE, Vol. 6603 of Lecture Notes in Computer Science, Springer, 2011, pp. 2–17. doi:10.1007/978-3-642-19811-3_2.

[13] F. He, X. Gao, B. Wang, L. Zhang, Leveraging weighted automata in compositional reasoning about concurrent probabilistic systems, in: S. K. Rajamani, D. Walker (Eds.), POPL, ACM, 2015, pp. 503–514. doi:10.1145/2676726.2676998.

[14] D. A. Peled, M. Y. Vardi, M. Yannakakis, Black box checking, J. Autom. Lang. Comb. 7 (2) (2002) 225–246. doi:10.25596/jalc-2002-225.

[15] A. Hagerer, H. Hungar, O. Niese, B. Steffen, Model generation by moderated regular extrapolation, in: R. Kutsche, H. Weber (Eds.), FASE, Vol. 2306 of Lecture Notes in Computer Science, Springer, 2002, pp. 80–95. doi:10.1007/3-540-45923-5_6.

[16] F. Wang, J. Wu, C. Huang, K. Chang, Evolving a test oracle in black-box testing, in: D. Giannakopoulou, F. Orejas (Eds.), FASE, Vol. 6603 of Lecture Notes in Computer Science, Springer, 2011, pp. 310–325. doi:10.1007/978-3-642-19811-3_22.

[17] R. Alur, P. Cerný, P. Madhusudan, W. Nam, Synthesis of interface specifications for java classes, in: J. Palsberg, M. Abadi (Eds.), POPL, ACM, 2005, pp. 98–109. doi:10.1145/1040305.1040314.

[18] F. Howar, D. Giannakopoulou, Z. Rakamaric, Hybrid learning: interface generation through static, dynamic, and symbolic analysis, in: M. Pezzè, M. Harman (Eds.), ISSTA, ACM, 2013, pp. 268–279. doi:10.1145/2483760.2483783.

[19] D. Giannakopoulou, Z. Rakamaric, V. Raman, Symbolic learning of component interfaces, in: A. Miné, D. Schmidt (Eds.), SAS, Vol. 7460 of Lecture Notes in Computer Science, Springer, 2012, pp. 248–264. doi:10.1007/978-3-642-33125-1_18.

[20] J. Sun, H. Xiao, Y. Liu, S. Lin, S. Qin, TLV: abstraction through testing, learning, and validation, in: E. D. Nitto, M. Harman, P. Heymans (Eds.), ESEC/FSE, ACM, 2015, pp. 698–709. doi:10.1145/2786805.2786817.

[21] D. Giannakopoulou, Z. Rakamaric, V. Raman, Symbolic learning of component interfaces 7460 (2012) 248–264. doi:10.1007/978-3-642-33125-1_18.

[22] M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, M. Tautschnig, Learning the language of error, in: B. Finkbeiner, G. Pu, L. Zhang (Eds.), ATVA, Vol. 9364 of Lecture Notes in Computer Science, Springer, 2015, pp. 114–130. doi:10.1007/978-3-319-24953-7_9.

[23] Y. Chen, C. Hsieh, O. Lengál, T. Lii, M. Tsai, B. Wang, F. Wang, PAC learning-based verification and model synthesis, in: L. K. Dillon, W. Visser, L. A. Williams (Eds.), ICSE, ACM, 2016, pp. 714–724. doi:10.1145/2884781.2884860.

[24] H. Xiao, J. Sun, Y. Liu, S. Lin, C. Sun, Tzuyu: Learning stateful typestates, in: E. Denney, T. Bultan, A. Zeller (Eds.), ASE, IEEE, 2013, pp. 432–442. doi:10.1109/ASE.2013.6693101.

[25] F. W. Vaandrager, Model learning, Commun. ACM 60 (2) (2017) 86–95. doi:10.1145/2967606.

[26] B. Bollig, J. Katoen, C. Kern, M. Leucker, D. Neider, D. R. Piegdon, libalf: The automata learning framework, in: T. Touili, B. Cook, P. B. Jackson (Eds.), CAV, Vol. 6174 of Lecture Notes in Computer Science, Springer, 2010, pp. 360–364. doi:10.1007/978-3-642-14295-6_32.

[27] The open-source learnlib - A framework for active automata learning, in: D. Kroening, C. S. Pasareanu (Eds.), CAV, Vol. 9206 of Lecture Notes in Computer Science, Springer, 2015, pp. 487–495. doi:10.1007/978-3-319-21690-4_32.

[28] F. Howar, B. Steffen, B. Jonsson, S. Cassel, Inferring canonical register automata, in: V. Kuncak, A. Rybalchenko (Eds.), VMCAI, Vol. 7148 of Lecture Notes in Computer Science, Springer, 2012, pp. 251–266. doi:10.1007/978-3-642-27940-9_17.

[29] M. Isberner, F. Howar, B. Steffen, Learning register automata: from languages to program structures, Mach. Learn. 96 (1-2) (2014) 65–98. doi:10.1007/s10994-013-5419-7.

[30] J. An, M. Chen, B. Zhan, N. Zhan, M. Zhang, Learning one-clock timed automata 12078 (2020) 444–462. doi:10.1007/978-3-030-45190-5_25.

[31] B. Alpern, F. B. Schneider, Recognizing safety and liveness, Distributed Comput. 2 (3) (1987) 117–126. doi:10.1007/BF01782772.

[32] M. Y. Vardi, P. Wolper, An automata-theoretic approach to automatic program verification, in: LICS, 1986, pp. 322–331.

[33] A. Pnueli, R. Rosner, On the synthesis of a reactive module, in: Conference Record of POPL 1989, ACM Press, 1989, pp. 179–190. doi:10.1145/75277.75293.

[34] C. S. Lee, N. D. Jones, A. M. Ben-Amram, The size-change principle for program termination, in: C. Hankin, D. Schmidt (Eds.), Conference Record of POPL 2001, ACM, 2001, pp. 81–92. doi:10.1145/360204.360210.

[35] O. Maler, A. Pnueli, On the learnability of infinitary regular sets, Inf. Comput. 118 (2) (1995) 316–326. doi:10.1006/inco.1995.1070.

[36] A. Farzan, Y. Chen, E. M. Clarke, Y. Tsay, B. Wang, Extending automated compositional verification to the full class of omega-regular languages, in: C. R. Ramakrishnan, J. Rehof (Eds.), TACAS, Vol. 4963 of Lecture Notes in Computer Science, Springer, 2008, pp. 2–17. doi:10.1007/978-3-540-78800-3_2.

[37] H. Calbrix, M. Nivat, A. Podelski, Ultimately periodic words of rational w-languages, in: S. D. Brookes, M. G. Main, A. Melton, M. W. Mislove, D. A. Schmidt (Eds.), Mathematical Foundations of Programming Semantics, Vol. 802 of Lecture Notes in Computer Science, Springer, 1993, pp. 554–566. doi:10.1007/3-540-58027-1_27.

[38] O. Maler, L. Staiger, On syntactic congruences for omega-languages, in: P. Enjalbert, A. Finkel, K. W. Wagner (Eds.), STACS, Vol. 665 of Lecture Notes in Computer Science, Springer, 1993, pp. 586–594. doi:10.1007/3-540-56503-5_58.

[39] M. Heizmann, J. Hoenicke, A. Podelski, Termination analysis by learning terminating programs, in: A. Biere, R. Bloem (Eds.), CAV, Vol. 8559 of Lecture Notes in Computer Science, Springer, 2014, pp. 797–813. doi:10.1007/978-3-319-08867-9_53.

[40] C. Courcoubetis, M. Yannakakis, The complexity of probabilistic verification, J. ACM 42 (4) (1995) 857–907. doi:10.1145/210332.210339.

[41] M. Isberner, F. Howar, B. Steffen, The TTT algorithm: A redundancy-free approach to active automata learning, in: B. Bonakdarpour, S. A. Smolka (Eds.), RV, Vol. 8734 of Lecture Notes in Computer Science, Springer, 2014, pp. 307–322. doi:10.1007/978-3-319-11164-3_26.

[42] Y. Li, X. Sun, A. Turrini, Y. Chen, J. Xu, ROLL 1.0: ω -regular language learning library, in: T. Vojnar, L. Zhang (Eds.), TACAS, Vol. 11427 of Lecture Notes in Computer Science, Springer, 2019, pp. 365–371. doi:10.1007/978-3-030-17462-0_23.

[43] Y. Tsay, M. Tsai, J. Chang, Y. Chang, Büchi store: An open repository of büchi automata, in: P. A. Abdulla, K. R. M. Leino (Eds.), TACAS, Vol. 6605 of Lecture Notes in Computer Science, Springer, 2011, pp. 262–266. doi:10.1007/978-3-642-19835-9_23.

[44] Y. Li, Y. Chen, L. Zhang, D. Liu, A novel learning algorithm for büchi automata based on family of dfas and classification trees, in: A. Legay, T. Margaria (Eds.), TACAS, Vol. 10205 of Lecture Notes in Computer Science, 2017, pp. 208–226. doi:10.1007/978-3-662-54577-5_12.

[45] M. Y. Vardi, Automatic verification of probabilistic concurrent finite-state programs, in: 26th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1985, pp. 327–338. doi:10.1109/SFCS.1985.12.

[46] S. Sickert, J. Esparza, S. Jaax, J. Kretínský, Limit-deterministic büchi automata for linear temporal logic, in: S. Chaudhuri, A. Farzan (Eds.), CAV, Vol. 9780 of Lecture Notes in Computer Science, Springer, 2016, pp. 312–332. doi:10.1007/978-3-319-41540-6_17.

[47] F. Blahoudek, M. Heizmann, S. Schewe, J. Strejcek, M. Tsai, Complementing semi-deterministic büchi automata, in: M. Chechik, J. Raskin (Eds.), TACAS, Vol. 9636 of Lecture Notes in Computer Science, Springer, 2016, pp. 770–787. doi:10.1007/978-3-662-49674-9_49.

[48] J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Longman Publishing Co., Inc., 2006.

[49] J. R. Büchi, On a decision method in restricted second order arithmetic, in: Int. Congress on Logic, Methodology and Philosophy of Science, 1962, pp. 1–11.

[50] C. Baier, J. Katoen, Principles of model checking, MIT Press, 2008.

[51] D. Angluin, D. Fisman, Regular omega-languages with an informative right congruence, in: A. Orlandini, M. Zimmermann (Eds.), GandALF 2018, Vol. 277 of EPTCS, 2018, pp. 265–279. doi:10.4204/EPTCS.277.19.

[52] A. Arnold, A syntactic congruence for rational omega-language, Theor. Comput. Sci. 39 (1985) 333–335. doi:10.1016/0304-3975(85)90148-3.

[53] O. Maler, L. Staiger, On syntactic congruences for omega-languages, in: P. Enjalbert, A. Finkel, K. W. Wagner (Eds.), STACS, Vol. 665 of Lecture Notes in Computer Science, Springer, 1993, pp. 586–594. doi:10.1007/

3-540-56503-5_58.

[54] D. Angluin, U. Boker, D. Fisman, Families of dfas as acceptors of omega-regular languages, in: P. Faliszewski, A. Muscholl, R. Niedermeier (Eds.), MFCS, Vol. 58 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 11:1–11:14. doi:10.4230/LIPIcs.MFCS.2016.11.

[55] P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, T. Vojnar, Simulation subsumption in ramsey-based büchi automata universality and inclusion testing, in: T. Touili, B. Cook, P. B. Jackson (Eds.), CAV, Vol. 6174 of Lecture Notes in Computer Science, Springer, 2010, pp. 132–147. doi:10.1007/978-3-642-14295-6_14.

[56] P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, T. Vojnar, Advanced ramsey-based büchi automata inclusion testing, in: J. Katoen, B. König (Eds.), CONCUR, Vol. 6901 of Lecture Notes in Computer Science, Springer, 2011, pp. 187–202. doi:10.1007/978-3-642-23217-6_13.

[57] W. Thomas, Automata on infinite objects, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, Elsevier and MIT Press, 1990, pp. 133–191. doi:10.1016/b978-0-444-88074-1.50009-3.

[58] D. Angluin, Queries and concept learning, Mach. Learn. 2 (4) (1987) 319–342. doi:10.1007/BF00116828.

[59] D. Angluin, T. Antonopoulos, D. Fisman, Query learning of derived ω-tree languages in polynomial time, Log. Methods Comput. Sci. 15 (3). doi:10.23638/LMCS-15(3:21)2019.

[60] J. Moerman, M. Sammartino, A. Silva, B. Klin, M. Szynwelski, Learning nominal automata, in: G. Castagna, A. D. Gordon (Eds.), POPL, ACM, 2017, pp. 613–625. doi:10.1145/3009837.3009879.

[61] D. Angluin, D. Fisman, Y. Shoval, Polynomial identification of ω-automata, in: A. Biere, D. Parker (Eds.), TACAS, Vol. 12079 of Lecture Notes in Computer Science, Springer, 2020, pp. 325–343. doi:10.1007/978-3-030-45237-7_20.